# Top-k Publish/Subscribe Matching Model Based on Sliding Window

**Pripužić, Krešimir**

UNIVERSITY OF ZAGREB
**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**
SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**Krešimir Pripužić**

# TOP-K PUBLISH/SUBSCRIBE MATCHING MODEL BASED ON SLIDING WINDOW

# MODEL USPOREDBE "K NAJBOLJIH" U SUSTAVIMA OBJAVI-PRETPLATI TEMELJEN NA KLIZEĆEM PROZORU

DOCTORAL THESIS
DOKTORSKA DISERTACIJA

Zagreb, 2010.

The doctoral dissertation has been completed at the Department of Telecommunications of the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia.

*Advisor*: Karl Aberer, Ph.D., professor
School of Computer and Communication Sciences,
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.

*Advisor*: Ivana Podnar Žarko, Ph.D., assistant professor
Faculty of Electrical Engineering and Computing,
University of Zagreb, Croatia.

The dissertation has 175 pages.
Dissertation number:

The dissertation evaluation committee:

1. Committee member: Karl Aberer, Ph.D., professor,
   School of Computer and Communication Sciences,
   École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
2. Committee member: Bojana Dalbelo-Bašić, Ph.D., professor,
   Faculty of Electrical Engineering and Computing,
   University of Zagreb, Croatia.
3. Committee chair: Ignac Lovrek, Ph.D., professor,
   Faculty of Electrical Engineering and Computing,
   University of Zagreb, Croatia.
4. Committee member: Robert Manger, Ph.D., professor,
   Faculty of Science,
   University of Zagreb, Croatia.
5. Committee member: Ivana Podnar Žarko, Ph.D., assistant professor,
   Faculty of Electrical Engineering and Computing,
   University of Zagreb, Croatia.

The dissertation defense committee:

1. Committee member: Karl Aberer, Ph.D., professor,
   School of Computer and Communication Sciences,
   École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
2. Committee member: Bojana Dalbelo-Bašić, Ph.D., professor,
   Faculty of Electrical Engineering and Computing,
   University of Zagreb, Croatia.
3. Committee chair: Ignac Lovrek, Ph.D., professor,
   Faculty of Electrical Engineering and Computing,
   University of Zagreb, Croatia.
4. Committee member: Robert Manger, Ph.D., professor,
   Faculty of Science,
   University of Zagreb, Croatia.
5. Committee member: Ivana Podnar Žarko, Ph.D., assistant professor,
   Faculty of Electrical Engineering and Computing,
   University of Zagreb, Croatia.

Date of dissertation defense: June 15$^{th}$, 2010

# Acknowledgments

A number of people have supported me during my PhD studies and work on this thesis. First of all, I would like to thank my advisor prof. Karl Aberer for his support, guidance, patience and brilliance in turning my preliminary research idea into a well defined and quite efficiently solved scientific problem. Then, I would like to thank my second advisor, prof. Ivana Podnar Žarko, for her diligence and patience in designing experiments, structuring the thesis, and in reading and improving dozens of incomprehensibly written pages filled with mathematical formulas and proofs. I would also like to thank prof. Ignac Lovrek, the chair of my dissertation evaluation committee, for managing complex administrative tasks stemming from the international nature of my dual mentorship.

The major ideas presented in this thesis have been developed during my one-year research visit at the Laboratoire de systèmes d'information répartis (LSIR), École Polytechnique Fédérale de Lausanne, Switzerland. I thank the members of LSIR for creating an inspiring research environment, and for making my stay in Lausanne a pleasant experience. Since this visit would not be possible without the scholarship received from the State Secretariat for Education and Research (ESKAS) of the Swiss Government, I would like to thank ESKAS for its financial support, and the Embassy of Switzerland in Croatia for the administrative work related to the visit.

I would also like to thank my colleagues Hrvoje Belani, Marin Vuković, Tomislav Grgić, Mirko Sužnjević and Igor Ljubi at the Department of Telecommunications, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, for taking some of my teaching obligations during the final phase of the thesis writeup.

For instilling in me the foundations of mathematics, programming and logical thinking, I would like to thank my gymnasium professors: prof. Ivan Cindrić, prof. Josip Čeple and prof. Nedeljko Begović.

And most importantly, I would like to thank my parents Blanka and Ivan, my brothers Matija and Dominik and my girlfriend Mirjana for their continuous support and encouragement through the entire duration of my PhD studies and lengthy work on this thesis.

# Abstract

The publish/subscribe communication paradigm is based on the delivery of publications to subscribers in accordance with their information needs expressed by a set of subscriptions. In this thesis we analyze the existing models of matching publications to subscriptions in publish/subscribe systems, and show that such models do not provide any means for controlling the number of delivered publications per subscription which is crucial to ensure the quality and performance of an entire publish/subscribe system.

In this thesis we formally define the Boolean matching model, a publish/subscribe matching model which is a generalization of the currently widely-used matching models, namely topic-based, content-based, and type-based. This matching model considers all matching publications equally relevant to a subscription. We also formally define the top-k/w matching model, a novel publish/subscribe matching model which enables a subscriber to control the number of publications it will receive per subscription within a predefined time period. In this matching model, each subscription defines an arbitrary and time-independent scoring function and parameters $k \in \mathbb{N}$ and window-size $w \in \mathbb{R}^+$ such that, at a point in time $t$, the parameter $k$ limits the number of delivered publications restricting it to the $k$ best scored publications that are published between points in time $t-w$ and $t$. Additionally, we formally specify publish/subscribe systems that are based on these two matching models.

In this thesis we propose novel algorithms for efficient processing of multiple top-k/w queries (i.e. top-k/w subscriptions) in resource-constrained environments publishing data objects (i.e. publications) at high rates. Existing solutions either assume processing environments with unbounded memory, or scale poorly for large values of parameter $k$ and window size $w$. Moreover, they are typically tailored to specific scoring functions, and thus we define a generic data stream processing model that is independent of data representation and scoring functions. Instead of maintaining the minimal set of potential top-k objects in a k-skyband, we propose a novel deterministic algorithm which periodically prunes the k-skyband to offer improved processing performance at the expense of an increased memory consumption. Furthermore, we present a probabilistic algorithm for random-order data streams which surpasses deterministic algorithms in terms of memory consumption and processing performance, however, it produces approximate answers to top-k/w queries. The introduced probabilistic criterion can further improve deterministic algo-

rithm implementations by maintaining a special buffer with recent objects, and such improved algorithms outperform their original implementations. Our complexity analysis and extensive comparative evaluation using both synthetic and real datasets show that the proposed algorithms significantly outperform the competing approaches in both memory consumption and processing efficiency.

Since scalability is an essential characteristic of publish/subscribe systems, special attention is given to the design of algorithms supporting the proposed top-k/w matching model which target large-scale publish/subscribe systems. For each of the commonly used routing strategies in both centralized and distributed environments we first explain how it can be adapted to support the proposed top-k/w matching model, then we define the system architecture comprising top-k/w processors and recent buffers, and finally explain how subscriptions and publications are routed in the system. We identify and evaluate routing strategies which are particularly well suited for large-scale top-k/w publish/subscribe systems. Additionally, we present and experimentally evaluate D-ZaLaPS, a distributed top-k/w publish/subscribe system which is based on a peer-to-peer structured overlay. The experimental evaluation shows that D-ZaLaPS is scalable both for the increasing number of peers and subscriptions.

Finally, using the case study with k-nearest neighbor subscriptions, we experimentally evaluate the top-k/w matching model and compare it with the existing Boolean matching model. This experimental evaluation shows that, assuming the subscribers are interested in the top-k publications in the sliding window, the top-k/w matching model can significantly reduce message overhead in the system when compared to the Boolean matching model.

# Contents

# List of Figures

# List of Tables

*We have reason to fear that the multitude of books which grows every day in a prodigious fashion will make the following centuries fall into a state as barbarous as that of the centuries that followed the fall of the Roman Empire.*
*Unless we try to prevent this danger by separating those books which we must throw out or leave in oblivion from those which one should save and within the latter between what is useful and what is not.*

Adrien Baillet (1649 – 1706)

# CHAPTER 1

# Introduction

For more than a decade, we have witnessed an exponential growth of the amount of newly created digital information. The IDC research [92, 93] shows that the digital universe—information that is either created, captured, or replicated in digital form—was 161 exabytes[1] in the year 2006, and 281 exabytes in the year 2007. Additionally, according to the same research, the compound annual growth rate of the digital universe between now and 2011 is expected to be almost 60%, which would result in nearly 1800 exabytes in the year 2011, or 10 times more than in the year 2006. This phenomenon is called the *information explosion* and is happening faster than it was initially predicted several years ago in the UC Berkeley study [133, 134].

The information explosion is not exclusive to digital information. For example, an analogous phenomenon is happening with scientific information since traditional scientific publishing in peer-reviewed journals is constantly increasing and there are no indications that the growth rate has decreased in the last 50 years, while new publishing channels such as conference proceedings are opening at the same time [123]. Additionally, as shown in [105], the traditional scientific publishing is expected to increase considerably in future.

Actually, the information explosion is not specific to the modern age, as a similar problem—the exponential growth of the amount of printed word—had occurred after the invention and spread of the printing press. For example, in the epigraph of this chapter we give a quote from the year 1685 by Adrien Baillet, a French scholar and critic, who warned about the overabundance of books in his time [31].

The information explosion would not pose such a large problem if we could store all digital information we create. However, during the year 2007 we have passed the point where this can be achived and by the year 2011, almost half of the digital universe will not be recorded on information storage mediums [93]. Therefore, for the first time, we are facing the situation where all created digital information cannot

[1] 1 exabyte = $1,000,000,000,000,000,000$ bytes = $10^{18}$ bytes = 1 billion gigabytes.

be stored. Of course, all created information does not have equal value to be stored for the future. For example, we can agree that a collection of private photos is much more important than downloaded RSS feeds or a collection of recipes.

Since all produced digital information cannot be stored, it is necessary to process such information before storing it to decide which actually has value to be stored. A similar processing task is also required to determine whether a user should be exposed to newly created information or not. Besides negative effects such as lost time and possible loss of money, the exposure to large quantities of information may produce even worse side-effects, such as the information overload which we discuss next.

A phenomenon highly-related to the information explosion is *information overload*. It is defined as a rapid decline in performance (in terms of adequate decision making) of an individual being exposed to too much information [80]. It has been shown that the performance of an individual raises positively up to a certain point with the amount of information he or she receives, but declines sharply if additional information is given beyond this point [59]. Information overload affects many areas of our everyday lives: We get an incredible amount of e-mail messages every day; we are often interrupted by instant messages and telephone calls; and we must also check social networking sites, news sources and other web sites on a daily basis, or even more often.

The information overload is particularly noticeable in science where the Internet allows timely dissemination of new research results, resulting in a tremendous increase of the quantity and diversity of easily available research information, especially in rapidly advancing fields [33]. Actually, scientific literature on the Internet can be treated as a massive, noisy, disorganized database from which researchers are trying to discover and extract useful knowledge, i.e. only those publications that may be relevant or interesting for them [33].

## 1.1 Paradigm Shift in Data Processing

Today we are witnessing a paradigm shift in data processing since we are moving away from the traditional *store-then-query* model of database management systems (DBMSs) toward the novel *query-then-store* model of emerging data stream processing systems (DSPSs) [94]. There are three main reasons for this shift. First, for many application scenarios it would be very difficult, if not impossible, to store all produced information in a database. Second, even if we could store it to databases, this would be unreasonable because usually only a small part would be accessed again. Third, this type of processing is generally slow since data has to be first stored in a database before we can start querying it.

For this reason, traditional databases are today typically used in applications requiring persistent data storage and support for complex one-time queries, where both the data and especially relations change less frequently than queries [98]. These databases are not designed for high rate of insertions, updates and deletions of streaming data objects, but rather for a high query rate.

On the other side, data stream processing systems are used in application scenarios which assume a high rate of data object insertions and mostly static and continuous queries that are executed on data objects upon each new object arrival. The novel query-then-store processing paradigm is vital for a num-

ber of applications, such as in-network data processing [166], on-line auctions [183], real-time decision support [57], network monitoring [96], sensor networks [7], financial analysis [208], fraud detection [86], on-line gaming [38], web management [112] and news feeds [132]. In this thesis we are interested in the latter systems, i.e. in the query-then-store paradigm.

## 1.2 Motivation

Nowadays, there is an emerging need for information filtering systems that will be able to process huge amounts of information for a large number of users, preferably in real-time, and effectively prevent information overload by supplying users only with useful information. This need has been recognized even 15 years ago [26], and today we need even larger Internet-scale information filtering systems. Information has to be personalized because in such an abundance of information, users do not want to receive information in which they are not interested. Additionally, in many usage scenarios, new information items are more valuable to users than old and thus the information has to be delivered to users immediately after being produced. For example, we can agree that information about a good offer on an auction site becomes worthless after the bidding closes. Finally, users want to receive only a limited amount of information to prevent information overload and this information has to be of top quality.

Three types of data processing systems are dealing with the problem of satisfying user information needs in real-time while preventing information overload, but neither is fully capable to accomplish this goal. These systems are web search engines, publish/subscribe systems (also called event-based systems) and data stream processing systems. In this thesis we present a hybrid publish/subscribe system model which uses concepts from all three research areas to achieve the given goal. Hereafter we briefly survey and point out shortcomings of the three types of systems.

### 1.2.1 Web Search Engines

A web search engine is a distributed information system which is primarily designed to search for textual web pages. Upon accepting an one-time user query, the search engine returns a list of ranked pages. In order to obtain a low average processing time, the search engine constantly crawls the web and builds an index using retrieved data. Since crawling has to cover a lot of dynamic web sites, there is no guarantee that the indexed data is up-to-date. The index is used in combination with a ranking algorithm to compute answers to queries.

Web search engines are based on the store-then-query paradigm since the building (and storing) of an index happens before querying. Obviously, they are not designed to be real-time systems since information first has to be crawled to even get a chance to be returned as an answer, and as already explained, this is a slow process. However, there are some recent research results in the direction of real-time web search [95].

Since every crawled page on the web is a potential answer to a query, search engines provide efficient filtering of huge amount of previously crawled data. On the other hand, due to the information explosion, the part of the web which is not indexed by search engines could be more than 500 times larger than the

indexed part [27]. In other words, most of the web content is not indexed, and thus cannot be discovered and accessed using web search engines.

A typical search engine supports a simple Boolean query language and its ranking algorithm is completely unknown to users, which can be an additional source of frustration for inexperienced Internet users [151]. As a matter of fact, a user will probably rank the same set of pages differently than the search engine, but this discussion is out of scope of this thesis. However, the wide acceptance and intuitiveness of web search is currently indisputable.

### 1.2.2 Publish/Subscribe Systems

In a typical publish/subscribe system there are *publishers* and *subscribers*. Publishers *publish* information content which they want to distribute among subscribers in messages called *publications*. Subscribers *subscribe* to information content in which they are interested by issuing messages called *subscriptions*. The *publish/subscribe service*, which can be *centralized* on a single network node or *distributed* over several network nodes, is located between publishers and subscribers, and is responsible for delivering published publications in a timely manner to only those subscribers whose subscriptions *match* these publications. Obviously, publish/subscribe systems are based on the query-then-store paradigm since publications are first processed by the publish/subscribe service, and then, if necessary, they will be stored by subscribers.

Every publish/subscribe system is distributed in its nature because it consists of a number of clients (i.e. publishers and subscribers) and a publish/subscribe service, which is the mediator that decouples publishers from subscribers. However, in literature, *centralized publish/subscribe systems* denote systems with a centralized publish/subscribe service, while *distributed publish/subscribe systems* denote systems in which this service is distributed over a number of network nodes which are loosely coupled. The main drawback of centralized publish/subscribe solutions is their poor scalability. For example, [29] shows that not a single enterprise service bus (ESB) product on the market sustains a high throughput when the number of subscribers increases. However, despite this apparent lack of scalability, these solutions are the best choice for small-scale publish/subscribe systems due to their simplicity. On the contrary, large-scale publish/subscribe systems are regularly implemented as distributed systems.

Distributed large-scale publish/subscribe systems are able to filter large amounts of published information in real-time, but despite of extensive research efforts done in the last 15 years, these systems are yet to be widely deployed. According to [175], the main reasons for their slow acceptance are the following: 1) complexity of the general matching problem, 2) system heterogeneity and 3) lack of wide-area deployments which delays the development of advanced solutions for real-life application scenarios. In this thesis we argue that there is another reason for the lack of adoption of these systems—an unpredictable number of matching publications with the currently employed *Boolean matching model*. Either too many or too few received publications may cause user dissatisfaction with a provided service, for example, in applications such as RSS news feeds, network monitoring, or advertisement dissemination. Moreover, in networks with limited resources such as MANETs or sensor networks, it is highly desirable to minimize and control network traffic.

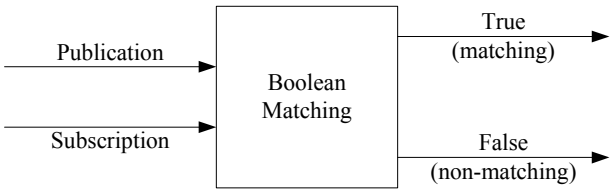



Figure 1.1: Matching in the Boolean publish/subscribe system.

In current publish/subscribe systems, subscription is a stateless Boolean function [146]: A decision whether to deliver a publication to a subscriber is made based on the result of the matching process comparing the publication and the subscriber's subscription as shown in Figure 1.1. The matching process depends only on the publication and subscription content, and does not take into account any additional information present in the system. This approach has the following drawbacks:

- the subscriber may receive too few publications,
- the subscriber may be overloaded with publications,
- there is no ranking function to compare different publications, and
- partial matching between subscriptions and publications is not supported.

As publication content is generally unknown in advance, it is impossible to predict the number of future publications matching an existing subscription. If a subscription is *too general*, a subscriber may receive too many publications. On the contrary, in case of an *overspecified* subscription, the subscriber may receive too few publications or none in the worst case. Thus, a subscriber has to specify an "ideal" subscription to receive an optimal number of matching publications. It is a sort of guessing, where even a slight change in subscription may result in a drastically different number of matching publications. In general, a user perceives the entire system through both the quantity and quality of received publications. Therefore, a large quantity of received publications will be considered as a sort of spam, while a system that delivers too few publications might be recognized as non-working. The number of received publications is crucial for the acceptance of an actual system by users even more if, for example, subscribes pay for each delivered publication matching subscriber information interest.

State-of-the-art publish/subscribe systems do not support publication ranking and all matching publications are considered equally relevant to a subscription. On one side, without publication ranking it is impossible to avoid the delivery of a large number of matching publications if a subscription is too general. On the other side, overspecified subscriptions may deliver too few publications unless partial matching is supported. Unfortunately, as a consequence of the complexity of the general matching problem, state-of-the-art systems do not support partial matching[2]between publications and subscriptions since they focus on designing fast matching algorithms [115].

[2]This problem has already been recognized in the publish/subscribe community and thus some authors propose approximate subscriptions based on the principles of fuzzy logic [128, 129, 8, 164, 42, 124].

### 1.2.3 Data Stream Processing Systems

A data stream is a continuous and possibly infinite sequence of data objects that arrive in an arbitrary order into a processing system to be processed in real-time [98]. In such a setting, a data object from an input data stream is matched against a set of continuous queries upon its arrival into a processing system, and is subsequently inserted into the output stream associated with a continuous query if the object matches the query.

Distributed data stream processing systems provide efficient processing of (both continuous and one-time) complex queries, but in contrast to distributed publish/subscribe systems, they are typically tightly coupled, platform dependent, difficult to deploy and maintain, and less scalable to the number of users [206]. Therefore, to achieve the given goal (i.e. to design a large-scale real-time information filtering system), we have to combine good characteristics of distributed publish/subscribe systems with both the complexity of continuous queries found in data stream processing systems and intuitiveness of web search.

## 1.3 Top-k/w Matching Model in Brief

In this thesis we present a new publish/subscribe matching model, named the *top-k/w matching model*. This matching model enables a subscriber to control the number of publications it will receive per subscription within a predefined time period. In this matching model, each subscription in a system defines 1) an arbitrary and time-independent scoring function and 2) the parameters $k \in \mathbb{N}$ and $w \in \mathbb{R}^+$. Unlike the Boolean matching model, this matching model is time-dependent because at each point in time $t$, the parameter $k$ limits the number of matching publications restricting it to the $k$ best scored among publications that are published between points in time $t - w$ and $t$. This interval is called the *time-based window* of size $w$. Since this interval is different at every different point in time $t$, we say that the window is *sliding* in time.

We consider this model to be quite intuitive for Internet users as it requires that each subscriber defines a scoring function[3]and parameters $k$ and $w$. Web users are already familiar with the concepts of query, ranking and top-$k$ since every web search engine works in this way. Moreover, the additional parameter $w$, as the subscription window size, is quite intuitive as well.

Top-k/w subscriptions are useful for a number of real world data stream applications where recent publications (i.e. data objects) are more important than older ones, and have been identified as one of the most important types of continuous queries [114]. We identify a number of possible application scenarios that include top-k/w query processing. For example, in a news/blog feed application, recent items are more important than old ones and hence RSS readers typically list items according to their time of appearance regardless of their content. State-of-the-art RSS readers and filters do not support top-k/w monitoring that would enable users to receive a limited number of items most relevant to their personal

[3]Please note that application developers can also define a scoring function per system, which additionally simplifies subscriptions. For example, in this thesis experiments we use Euclidean distance as a scoring function, for which subscriptions are composed of an attribute space point and parameters k and w.

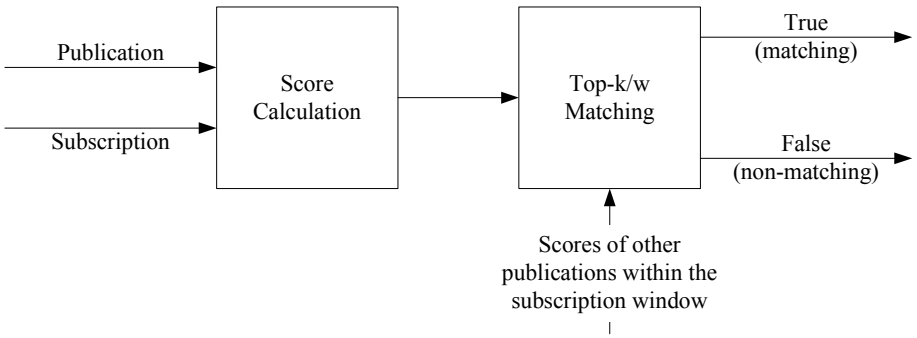



Figure 1.2: Matching in the top-k/w publish/subscribe system.

interest in real time. Existing RSS filters support exact-match queries where an item matches a continuous query only in case it contains all keywords from the query [180]. The next possible application scenario is environmental monitoring with a large sensor network used for measuring environmental conditions to detect and locate particular events such as early forest fire detection [170], pollution detection, animal tracking, etc. Another possible application scenario is monitoring computer [96] and telephone networks [52] to detect errors or suspicious events signaling network intrusion, misuse, or rule violation among huge quantities of data. Real-time analysis of financial data such as stock trading, cash flows and credit card transactions is another possible application area [208]. As the last scenario, consider advertisements on an arbitrary market where a user may specify properties of an ”ideal” product he/she is interested in, while the system would inform the user in real-time about newly appearing advertisements that are most similar to his/her ”ideal” product. Furthermore, the user could specify the number of advertisements he/she is willing to receive per day.

In this model, the quantity of received publications does not depend on the number of published publications, but on parameters $k$ and $w$. Obviously, the quality of received publications will depend on the scores of published publications with respect to a defined subscription, but statistically the quality will probably be proportional to the number of published publications. Please note that the size of a subscription window may also be defined as the number of publications competing simultaneously for a position among the top-k publications. This type of window is called the *number-based window*, and when it is used in a top-k/w model, the quantity of received publications depends on the number of published publications. The matching process in the top-k/w model depends both on the publication score and scores of other publications with respect to the same subscription as depicted in Figure 1.2.

Efficient processing of top-k/w subscriptions is challenging because, even though a publication is not a top-k publication at the time when it is published, it might become one in future, and therefore a set of potential top-k publications within the sliding window has to be stored in memory. For the efficient processing of top-k/w subscriptions it is imperative to remove from memory every publication which is detected as unable to become top-k publication in future.

**Example 1.1.** Figure 1.3 shows the processing of publications for a top-k/w subscription defining parameters $k = 2$ and number-based window $w = 5$. At each point in time a publication older than $w = 5$ (the crossed one) is dropped from the window, while a new one (the gray one) is added to it. At $t = 6$

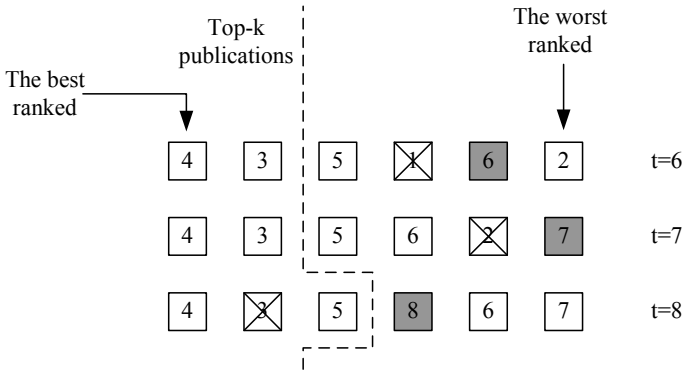



Figure 1.3: Processing publications for a top-k/w subscription.

the publication published at $t = 1$ is dropped, and a new one is added to the window. From the presented sequence of events, we can see that the publication published at $t = 5$ becomes a top-2 publication at $t = 8$ although it was not among the top-2 publications when it was published.

## 1.4 Contributions

This thesis consists of three parts: 1) the specification and evaluation of Boolean and top-k/w publish subscribe systems, 2) the formal model and efficient algorithms for top-k/w processing over data streams and 3) implementations and evaluations of centralized and distributed top-k/w publish/subscribe systems.

In the first, theoretical, part of this thesis, we formally define two publish/subscribe matching models: the Boolean and top-k/w model. After that, we formally specify two publish/subscribe systems based on these matching models. Finally, we experimentally evaluate the top-k/w matching model and compare its performances with the Boolean matching model. The contributions of the first part of this thesis are summarized as follows:

1. We provide an analysis of the shortcomings of the existing matching models in publish/subscribe systems. This analysis, together with results of an experimental evaluation using sliding-window k-NN subscriptions (i.e. queries) as a case study, shows that these matching models do not provide any means of controlling the number of delivered publications per subscription.

2. We present a formal definition of the Boolean matching model, a publish/subscribe matching model which is a generalization of currently widely-used matching models, namely topic-based, content-based and type-based.

3. We give a formal specification of the Boolean system in Metric Interval Temporal Logic (MITL) [13], a publish/subscribe system which is based on the Boolean matching model.

4. We present a formal definition of the top-k/w matching model with time-based windows in MITL. This is a novel publish/subscribe matching model which enables a subscriber to control the number of publications it will receive per subscription within a predefined time period.

5. We give a formal definition of the top-k/w matching model with number-based windows in set theory [34]. This is a novel publish/subscribe matching model which controls the number of matching publications for a subscription per predefined number of published publications.

6. We present a formal specification of the top-k/w system in MITL, a publish/subscribe system which is based on the top-k/w matching model with time-based windows. We also give proof that every Boolean system is just a special case of a corresponding top-k/w system.

In the second part of the thesis we formally define a generic top-k/w processing model for data streams. After that, we propose novel algorithms for efficient top-k/w processing over data streams, and finally, we provide an extensive experimental evaluation and complexity analysis of top-k/w processing algorithms. The contributions of the second part of this thesis are summarized as follows:

7. We present a formal generic model for processing top-k/w queries over certain data streams which is independent of data representation and applied scoring function.

8. We propose the Relaxed Candidate Pruning Algorithm (RA), a novel deterministic top-k/w processing algorithm with periodical pruning of non-prospective top-k data objects.

9. We propose the Probabilistic Candidate Pruning Algorithm (PA), a novel probabilistic top-k/w processing algorithm for processing random-order data streams.

10. We extend the implementation of SA[4] and RA with a PA-based query filter (PF) which utilizes a buffer of most recent data objects to further improve performances of deterministic top-k/w algorithms.

11. We present the complexity analysis and results of an extensive experimental evaluation of the listed top-k/w processing algorithms using sliding-window k-NN queries as a case study. Our results show that our deterministic implementations with PF significantly outperform the competitive deterministic approaches, and that PA significantly outperforms all of the deterministic approaches while introducing an acceptable and controllable probability of error.

In the third part of this thesis, we analyze currently used routing strategies in both centralized and distributed publish/subscribe systems, and adapt them to the specifics of top-k/w publish/subscribe systems. We present and experimentally evaluate a centralized and distributed version of the top-k/w system. The contributions of the third part of this thesis are summarized as follows:

12. For each of the commonly used routing strategies in (both centralized and distributed) Boolean publish/subscribe systems we propose the changes necessary to apply this strategy in top-k/w systems. We also show that 2 out of 3 centralized and 4 out of 6 distributed routing strategies are particularly well suited for top-k/w systems since they do not introduce any significant additional communication overhead in these systems when compared to the equivalent Boolean systems.

[4]The Strict Candidate Pruning Algorithm (SA) is an extended version of a deterministic top-k/w processing algorithm originally proposed in [32]. This algorithm continuously maintains a minimal set of top-k data objects in a k-skyband.

13. We present the results of an experimental evaluation of centralized Boolean and top-k/w systems using sliding-window k-NN subscriptions as a case study. This experimental evaluation shows that, assuming the subscribers are interested in the top-k publications in the sliding window, the centralized top-k/w systems can significantly reduce message overhead when compared to the equivalent Boolean systems.

14. We present D-ZaLaPS, a distributed top-k/w publish/subscribe system which is built on top of Content Addressable Network (CAN) [178]. Additionally, we report the results of an experimental evaluation of D-ZaLaPS for both Boolean and top-k/w subscriptions using sliding-window k-NN subscriptions as a case study. This experimental evaluation shows that D-ZaLaPS is scalable for both increasing number of peers and subscriptions, and that top-k/w subscriptions can significantly reduce message overhead when compared to Boolean subscriptions assuming the subscribers are interested in the top-k publications in the sliding window.

## 1.5 Summary of Related Work

Formal specification of Boolean publish/subscribe systems has been introduced in [89, 144], where the authors used Linear-time Temporal Logic (LTL) [136]. Afterwards, the same group of authors first extended their formal specification with message completeness guarantees, and then additionally updated it in [147]. However, our formal specification is more general since it supports finite processing delays in the system and covers all possible use cases.

Although data stream processing has been a particularly active research area in the last years [52, 97, 4, 91], solutions for efficient processing of top-k/w queries are still lacking. The existing papers [117, 141, 71, 143, 32] do not offer a generalized model for top-k/w processing, but rather focus on special top-k problems (e.g. k-NN) and specific data representations. The algorithm presented in [32] is most relevant to our work and serves as the basis for SA. The algorithms presented in [141, 143] are defined for top-k/w queries using aggregation and distance scoring functions, respectively. Contrary to our approach, these algorithms store in memory all data objects within the window, which is inefficient and unfeasible in the case of data sources with high streaming rates. The approximate algorithm presented in [117] is most relevant to our PA, however it is also tailored to k-NN queries.

In the area of distributed publish/subscribe systems, the most relevant work to ours is [111] which introduces a new type of stateful subscription called the parametrized subscription. According to their work, we can regard the copies of an original top-k/w subscription in the network as parametrized subscriptions with subscription threshold as the only parameter. The authors discuss one of the possible routing strategies in a centralized and distributed case. Additionally, similar systems to our D-ZaLaPS are Mehdoot [102] and [55] since they also use CAN structured peer-to-peer overlay [178], but consider only Boolean subscriptions.

The idea to rank publications in publish/subscribe systems according to a subscription has been developed in parallel in our [172] and the following two papers [135, 76]. However, the other authors did not recognize the importance of sliding-window in publish/subscribe setting. In [76] the authors

present an approach in which a static time of expiration is associated with each subscription in the system, and afterwards, the same group of authors introduced two interesting concepts in publish/subscribe systems: publication freshness [75] and subscription novelty [70]. Publication freshness is modeled by linear decrease of publication scores in time, while subscription novelty is described by increasing scores of incoming publications for subscriptions that have rarely been satisfied in the past. Additionally, this group of authors presented PerfSIENA [77], an implementation of their ranking mechanism in SIENA [45] based on user preference. In the latter paper, the authors consider the top-k/w matching model, but do not focus on its efficient implementation, which is the main contribution of this thesis. Similarly, the top-k/w matching model is examined in [131], but the authors mainly focus on the quality of service issues.

## 1.6 Structure of the Thesis

The thesis is organized as follows. Chapter 2 formally specifies the Boolean and top-k/w matching model and the corresponding publish/subscribe systems, and discusses the difference between these two publish/subscribe systems. Chapter 3 presents a formal top-k/w data stream processing model and gives a formal explanation of different issues affecting the practical implementation of the processing model, and also presents an experimental evaluation of the several top-k/w processing algorithms. Chapter 4 defines and analyzes Boolean and top-k/w subscriptions, and also discuss some very important conceptual differences between them. Chapter 5 presents three commonly used routing strategies for centralized Boolean publish/subscribe systems, adapts these strategies to centralized top-k/w publish/subscribe systems, and experimentally evaluates the performance of Boolean and top-k/w centralized publish/subscribe systems. Chapter 6 presents six commonly used routing strategies for distributed Boolean publish/subscribe systems, adapts these strategies to distributed top-k/w publish/subscribe systems, and presents and experimentally evaluates D-ZaLaPS, a distributed top-k/w publish/subscribe system which is built on top of a structured peer-to-peer overlay. Chapter 7 summarizes the contributions of this thesis and gives guidelines for future work.

# CHAPTER 2

## Formal Specification of Publish/Subscribe Systems

In this chapter we give formal specifications of two different publish/subscribe systems using temporal logic. These systems do not make a distinction between publishers and subscribers considering them as clients which can simultaneously act as both publishers and subscribers. The difference between the presented systems is in the used matching model. The first system, named *Boolean publish/subscribe system*, uses the Boolean matching model, which is a generalization of the three most common matching models: topic-based, content-based and type-based models. The matching in this system is based on an arbitrary and time-independent Boolean matching function, which is defined by each subscription in a system. On the contrary, the *top-k/w publish/subscribe system* uses a novel matching model, named the *top-k/w matching model*. In this matching model, each subscription in a system defines 1) an arbitrary and time-independent scoring function and 2) the parameters $k \in \mathbb{N}$ and $w \in \mathbb{R}^+$. As opposed to the Boolean matching model, this matching model is time-dependent because at each point in time $t$, the parameter $k$ limits the number of matching publications restricting it to the $k$ best scored among publications that are published between points in time $t - w$ and $t$. This interval is called the *time-based window* of size $w$. Since this interval is different at every different point in time $t$, we say that the subscription window is *sliding* in time.

In this chapter we formalize the top-k/w system with time-based sliding windows using temporal logic. On the contrary, the top-k/w system with count-based sliding windows cannot be formalized in this way, and for its formalization we have to use set theory [34]. For completeness of this thesis, in Chapter 3 we formalize count-based sliding windows using set theory. As publish/subscribe systems are event-based systems for which the time order of events is extremely important, temporal logic specification is much more appropriate than the corresponding set theory specification. For this reason, in this chapter we give a formal specification of the top-k/w system in temporal logic.

The rest of this chapter is organized as follows. In Section 2.1 we briefly define temporal logic which we use for the formal specification of the publish/subscribe systems in the rest of this chapter. After

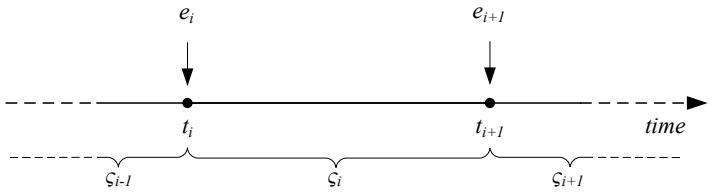



Figure 2.1: The transition of a real-time system.

that, we first formally specify the Boolean publish/subscribe system in Section 2.2, and then we formally specify the top-k/w system in Section 2.3. In Section 2.4 we focus our discussion on the difference between these two publish/subscribe systems. Finally, we conclude this chapter with an overview of the related work in Section 2.5.

## 2.1 Formal Background

An assignment of values to variables in a system is called the *system state* [122]. It is well known that individual behavior of a system can be described as an *infinite sequence of states* [136] which is called the *system trace*:

$$\sigma = \varsigma_0, \varsigma_1, \ldots, \varsigma_i, \ldots \tag{2.1}$$

where $\varsigma_0$ is the *initial state* of the system, where $i \in \mathbb{N}_0$ and where these states are ordered by their ascending times of occurrence. In other words, each execution of the system results in an infinite sequence of states. An event makes the *transition* from a previous state to a next state:

$$\varsigma_0 \xrightarrow{e_1} \varsigma_1 \xrightarrow{e_2} \varsigma_2, \ldots, \varsigma_{i-1} \xrightarrow{e_i} \varsigma_i \xrightarrow{e_{i+1}} \varsigma_{i+1} \cdots \tag{2.2}$$

The system does not change its state between two consecutive events. For example, event $e_i$ in Figure 2.1 makes the transition of a system from state $\varsigma_{i-1}$ to state $\varsigma_i$. Additionally, this system is in state $\varsigma_i$ during the period between events $e_i$ and $e_{i+1}$. Therefore, each state is uniquely determined by an initial state $\varsigma_0$ and the sequence of events that occurred before $\varsigma_i$. For example, in expression (2.2) state $\varsigma_2$ is completely determined by initial state $\varsigma_0$, and events $e_1$ and $e_2$.

We assume that two events cannot happen at the same point in time which allows us ordering of events in time by their time of occurrence.

An alternative way to describe the individual behavior of a system is an infinite and ordered sequence of events, which is called a *word*[1] over an *alphabet* $\alpha$ of events in the system:

$$\omega = a_1, a_2, \ldots, a_i, \ldots \tag{2.3}$$

where each event $e_i$ is represented by a *letter* $a_i$ from alphabet $\alpha$, and where the letters are ordered in time identically as their corresponding events. It is important to notice that we have to know the initial state of

[1] Term *word* is borrowed from the theory of timed automata [12].

a system to fully describe its behavior with a word $\omega$. However, sometimes we need to know the actual points in time at which events are occurring because their ordering in time is insufficient. For example, in Figure 2.1 we see exact points in time at which events are occurring.

**Definition 2.1** (Infinite Timed Word)**.** Let $\omega = a_1, a_2, \ldots$ be a word over an alphabet $\alpha$ of events in a system, and let $\zeta = t_1, t_2, \ldots$ be an infinite time sequence of points in time satisfying the following properties

$$\textit{(Strict-monotonocity)}\ \forall i \in \mathbb{N} : t_i < t_{i+1}, \text{ and} \tag{2.4}$$

$$\textit{(Progressiveness)}\quad \forall t \in \mathbb{R}^+, \exists i \in \mathbb{N} : t_i > t. \tag{2.5}$$

An infinite timed word $\tau = (\omega, \zeta)$ is an element of $(\alpha \times \mathbb{R}^+)$ of the form

$$(a_1, t_1), (a_2, t_2), \ldots, (a_i, t_i), \ldots. \tag{2.6}$$

An important distinction among real-time system models is whether one assumes that the system of interest is observed at every point in time, leading to an *interval-based semantics*, or whether one only sees a (possibly countably infinite) sequence of snapshots of the system, leading to *point-based semantics* [156]. In the latter semantics temporal formulas are interpreted only at specific time points when an event happens, whereas in the former they may be interpreted at some arbitrary time points between event occurrences as well. These semantics are also known as *pointwise* and *continuous*, respectively [169, 78].

For the formal specification of publish/subscribe systems in this chapter we use Metric Interval Temporal Logic (MITL) [13] with interval-based semantics and past operators [14]. MITL is a fragment of Metric Temporal Logic (MTL) in which temporal operators may only be constrained by non-singular intervals, whereas in MTL they can be constrained by both singular[2] and non-singular intervals [118]. It is proven that MTL is undecidable in interval-based semantics [15, 13, 109], while MITL is decidable in this semantics [13].

To provide the complete presentation in this chapter, hereafter we define MITL. We first define its grammar, then we define two basic temporal operators, and after that we define other (i.e. derived) temporal operators.

**Definition 2.2** (MITL Grammar)**.** The MITL formulas over an alphabet $\alpha$ of events in a system are inductively defined by the following grammar

$$\phi \stackrel{\text{def}}{=} a | \neg\phi | \phi_1 \wedge \phi_2 | \phi_1 \, \widehat{\mathcal{U}}_I \, \phi_2 | \phi_1 \, \widehat{\mathcal{S}}_I \, \phi_2, \tag{2.7}$$

where $a \in \alpha$ is any letter and $I$ is a non-singular interval with end-points which are either $\infty$ or elements of $\mathbb{R}^+$.

[2] A singular interval is the single point in time.

**Definition 2.3** (MITL Satisfaction Relation)**.** For an MITL-formula $\phi$, a timed word $\tau = (\omega, \zeta)$, and a point in time $t \in \mathbb{R}^+$, the satisfaction relation $(\tau, t) \vDash \phi$ is inductively defined as follows

$$(\tau, t) \vDash a \Leftrightarrow \exists i : t_i = t \wedge e_i = a, \tag{2.8}$$

$$(\tau, t) \vDash \neg\phi \Leftrightarrow (\tau, t) \nvDash \phi, \tag{2.9}$$

$$(\tau, t) \vDash \phi_1 \vee \phi_2 \Leftrightarrow (\tau, t) \vDash \phi_1 \vee (\tau, t) \vDash \phi_2, \tag{2.10}$$

$$\begin{aligned}(\tau, t) \vDash \phi_1 \,\widehat{\mathcal{U}}_I\, \phi_2 \Leftrightarrow \exists t' : \{&t < t' \wedge t' - t \in I \wedge (\tau, t') \vDash \phi_2 \wedge \\ &[\forall t'' : t < t'' < t' \Rightarrow (\tau, t'') \vDash \phi_1]\},\ \text{and}\end{aligned} \tag{2.11}$$

$$\begin{aligned}(\tau, t) \vDash \phi_1 \,\widehat{\mathcal{S}}_I\, \phi_2 \Leftrightarrow \exists t' : \{&0 < t' < t \wedge t - t' \in I \wedge (\tau, t') \vDash \phi_2 \wedge \\ &[\forall t'' : t' < t'' < t \Rightarrow (\tau, t'') \vDash \phi_1]\}.\end{aligned} \tag{2.12}$$

Operators $\widehat{\mathcal{U}}_I$ and $\widehat{\mathcal{S}}_I$ are called *strict until* and *strict since*, respectively. The former is a future, while the latter is a past temporal operator. Using these two operators we further define the derived temporal operators:

$$\text{(Strict eventually)} \qquad \widehat{\Diamond}_I \phi \stackrel{\text{def}}{=} \mathit{true}\, \widehat{\mathcal{U}}_I \phi, \tag{2.13}$$

$$\text{(Strict once)} \qquad \widehat{\blacklozenge}_I \phi \stackrel{\text{def}}{=} \mathit{true}\, \widehat{\mathcal{S}}_I \phi, \tag{2.14}$$

$$\text{(Strict always)} \qquad \widehat{\Box}_I \phi \stackrel{\text{def}}{=} \neg\widehat{\Diamond}_I \neg\phi, \tag{2.15}$$

$$\text{(Strict always in the past)} \qquad \widehat{\blacksquare}_I \phi \stackrel{\text{def}}{=} \neg\widehat{\blacklozenge}_I \neg\phi, \tag{2.16}$$

$$\text{(Strict unless)} \qquad \phi_1 \,\widehat{\mathcal{W}}_I\, \phi_2 \stackrel{\text{def}}{=} \phi_1 \,\widehat{\mathcal{U}}_I\, \phi_2 \vee \widehat{\Box}_I \phi_1,\ \text{and} \tag{2.17}$$

$$\text{(Strict back-to)} \qquad \phi_1 \,\widehat{\mathcal{B}}_I\, \phi_2 \stackrel{\text{def}}{=} \phi_1 \,\widehat{\mathcal{S}}_I\, \phi_2 \vee \widehat{\blacksquare}_I \phi_1. \tag{2.18}$$

Operators *strict eventually*, *strict always* and *strict unless* are future, while *strict once*, *strict always in the past* and *strict back-to* are past temporal operators. All previously defined operators do not include present and, as their names imply, they are strict (irreflexive) in their (first) argument [13, 136, 25]. We define the following non-strict versions of these operators:

$$\text{(Until)} \qquad \phi_1 \,\mathcal{U}_I\, \phi_2 \stackrel{\text{def}}{=} \phi_2 \vee (\phi_1 \wedge \phi_1 \,\widehat{\mathcal{U}}_I\, \phi_2), \tag{2.19}$$

$$\text{(Since)} \qquad \phi_1 \,\mathcal{S}_I\, \phi_2 \stackrel{\text{def}}{=} \phi_2 \vee (\phi_1 \wedge \phi_1 \,\widehat{\mathcal{S}}_I\, \phi_2), \tag{2.20}$$

$$\text{(Unless)} \qquad \phi_1 \,\mathcal{W}_I\, \phi_2 \stackrel{\text{def}}{=} \phi_2 \vee (\phi_1 \wedge \phi_1 \,\widehat{\mathcal{W}}_I\, \phi_2), \tag{2.21}$$

$$\text{(Back-to)} \qquad \phi_1 \,\mathcal{B}_I\, \phi_2 \stackrel{\text{def}}{=} \phi_2 \vee (\phi_1 \wedge \phi_1 \,\widehat{\mathcal{B}}_I\, \phi_2), \tag{2.22}$$

$$\text{(Eventually)} \qquad \Diamond_I \phi \stackrel{\text{def}}{=} \phi \vee \widehat{\Diamond}_I \phi, \tag{2.23}$$

$$\text{(Once)} \qquad \blacklozenge_I \phi \stackrel{\text{def}}{=} \phi \vee \widehat{\blacklozenge}_I \phi, \tag{2.24}$$

$$\text{(Always)} \qquad \Box_I \phi \stackrel{\text{def}}{=} \phi \wedge \widehat{\Box}_I \phi,\ \text{and} \tag{2.25}$$

$$\text{(Always in the past)} \qquad \blacksquare_I \phi \stackrel{\text{def}}{=} \phi \wedge \widehat{\blacksquare}_I \phi. \tag{2.26}$$

Although MITL prohibits the use of singular intervals, some temporal formulas with singular intervals

can be expressed in MITL. For example, in Section 2.3 we use the formula $\Box\{p \rightarrow [(\neg q)\mathcal{U}_{=t}q]\}$ which express the requirement that "for every $p$-state the next following $q$-state occurs after precisely $t$ time units". This formula is a valid MITL formula because in [13] it is proven that it can be expressed in MITL as follows

$$\Box\{p \rightarrow [(\neg q)\widehat{\mathcal{U}}_{=t}q]\} = \Box\{p \rightarrow [(\Box_{(0,t)}\neg q) \wedge (\Diamond_{(0,t]}q)]\}. \tag{2.27}$$

The interval $(0, \infty)$ is usually suppressed as a subscript, and such versions of temporal operators are called *unbounded temporal operators*. For example, the unbounded version of operator $\widehat{\mathcal{U}}_I$ is $\widehat{\mathcal{U}}$. The temporal logic with strictly unbounded operators is called Linear-time Temporal Logic (LTL) [136]. The same as in [13], we use intuitive pseudo-arithmetic expressions to denote intervals. For example, the following expressions $\leq b$ and $> a$ denote the intervals $(0, b]$ and $(a, \infty)$, respectively.

In this section we have presented a brief introduction to temporal logics, with focus on MITL and on the most important concepts that are necessary for understanding the following sections in this chapter. A reader interested in the subject of formal specification of real-time systems using temporal logic is further referred to Section 2.5.2 and the following textbooks [136], [25] and [119].

## 2.2 Boolean Publish/Subscribe System

In this section we formally specify the Boolean publish/subscribe system (*Boolean system*) using MITL. This is the currently most used publish/subscribe system. We use this system as a referent system in comparison with the top-k/w publish/subscribe system which we formally specify in Section 2.3.

### 2.2.1 Structural View

We define a triple $\boldsymbol{B} = (\boldsymbol{C}, \boldsymbol{P}, \boldsymbol{S})$, where $\boldsymbol{C}$ is a finite set of clients, $\boldsymbol{P}$ is a finite set of publications, and $\boldsymbol{S}$ is a finite set of subscriptions in a Boolean system. $\boldsymbol{B}$ gives the *structural view* of a Boolean system and determines the boundaries of the system state space, because it defines the type and number of entities that can exist in this system. A client $c \in \boldsymbol{C}$ may publish publications from $\boldsymbol{P}$ and activate or cancel subscriptions from $\boldsymbol{S}$. The following three system variables are defined for each client $c \in \boldsymbol{C}$ in a Boolean system:

- a finite set $P_c^P \subseteq \boldsymbol{P}$ of *published publications* (i.e. all publications which a client $c$ has published previously),
- a finite set $P_c^R \subseteq \boldsymbol{P}$ of *received publications* (i.e. all publications which a client $c$ has received previously), and
- a finite set $S_c^A \subseteq \boldsymbol{S}$ of *active subscriptions* (i.e. all subscriptions to which a client $c$ is subscribed to, and has still is not unsubscribed from).

**Definition 2.4** (System Variables)**.** For each client $c_i \in \boldsymbol{C}$ in a Boolean system, let $V_i \stackrel{\text{def}}{=} \{P_{c_i}^P, P_{c_i}^R, S_{c_i}^A\}$

denote the set of its variables. Then a finite set $\boldsymbol{V}$ of all variables in this system is defined as

$$\boldsymbol{V} \stackrel{\text{def}}{=} \{V_1, V_2, \ldots, V_{|\boldsymbol{C}|}\}, \tag{2.28}$$

which has the following cardinality $|\boldsymbol{V}| = 3 \cdot |\boldsymbol{C}|$.

For each client in a Boolean system we assume that its sets of published publications, received publications and active subscriptions are empty in the *initial state* of the Boolean system.

**Definition 2.5** (Initial State)**.** We define the initial state of a Boolean system as follows

$$\forall c \in \boldsymbol{C} : P_c^P(t=0) \stackrel{\text{def}}{=} \emptyset, P_c^R(t=0) \stackrel{\text{def}}{=} \emptyset, S_c^A(t=0) \stackrel{\text{def}}{=} \emptyset. \tag{2.29}$$

### 2.2.2 Behavioral View

The behavioral view of a Boolean system defines types of events that can occur in the system. The following events can occur: 1) *publish*, 2) *subscribe*, 3) *unsubscribe* and 4) *notify*. An event *publish* denotes the publishing of a new publication. Events *subscribe* and *unsubscribe* denote the activation and cancellation of a client subscription, respectively. The delivery of a matching publication to a subscribed client is denoted by an event *notify*. These four events determine the alphabet of events in a Boolean system, where each possible event in the system is represented by a single letter of this alphabet.

**Definition 2.6** (Alphabet)**.** Let $\boldsymbol{C}$ be a finite set of clients, let $\boldsymbol{P}$ be a finite set of publications, and let $\boldsymbol{S}$ be a finite set of subscriptions in a Boolean system. An alphabet $\alpha$ of the possible events in this system consists of the following letters:

$$\begin{aligned}
&\forall c \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \text{publish}(c,p) \in \alpha,\\
&\forall c \in \boldsymbol{C}, \forall s \in \boldsymbol{S} : \text{subscribe}(c,s) \in \alpha,\\
&\forall c \in \boldsymbol{C}, \forall s \in \boldsymbol{S} : \text{unsubscribe}(c,s) \in \alpha, \text{ and}\\
&\forall c \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \text{notify}(c,p) \in \alpha,
\end{aligned} \tag{2.30}$$

and thus has the following cardinality $|\alpha| = |\boldsymbol{C}||\boldsymbol{P}| + |\boldsymbol{C}||\boldsymbol{S}| + |\boldsymbol{C}||\boldsymbol{S}| + |\boldsymbol{C}||\boldsymbol{P}| = 2 \cdot |\boldsymbol{C}|(|\boldsymbol{P}| + |\boldsymbol{S}|)$.

In the rest of this subsection we define changes that an occurring event causes to the system variables of a Boolean system. Each event makes a transition of a Boolean system to a new state by making an almost instant change to its system variables. Between each two consecutive events, system variables of a Boolean system do not change their values and thus the system stays in one state during this interval. For example, in Figure 2.1 we see that the values of system variables are constant in the interval $t = (t_i, t_{i+1}]$, and consequently this system stays in state $\sigma_i$ during the whole interval.

**Definition 2.7** (Publish)**.** Let $c \in \boldsymbol{C}$ be a client, and let $p \in \boldsymbol{P}$ be a publication in a Boolean system. We define an event publish as

$$\text{publish}(c,p) \stackrel{\text{def}}{=} \{P_c^P(t+\varepsilon) = P_c^P(t) \cup p\}, \tag{2.31}$$

where $t$ is a time point when publish occurs, $\varepsilon$ is a small positive infinitesimal quantity of time, $P_c^P(t)$ is the set of publications that $c$ has published before $t$, and $P_c^P(t+\varepsilon)$ is the set of publications which $c$ published up to $t$ and including $t$.

**Definition 2.8** (Subscribe)**.** Let $c \in \boldsymbol{C}$ be a client, and let $s \in \boldsymbol{S}$ be a subscription in a Boolean system. We define an event subscribe as

$$\text{subscribe}(c,s) \stackrel{\text{def}}{=} \{S_c^A(t+\varepsilon) = S_c^A(t) \cup s\}, \tag{2.32}$$

where $t$ is a point in time when subscribe occurs, $\varepsilon$ is a small positive infinitesimal quantity of time, $S_c^A(t)$ is the set of active subscriptions of $c$ before subscribe occurs, and $S_c^A(t+\varepsilon)$ is the set of active subscriptions of $c$ after subscribe occurs.

**Definition 2.9** (Unsubscribe)**.** Let $c \in \boldsymbol{C}$ be a client, and let $s \in \boldsymbol{S}$ be a subscription in a Boolean system. We define an event unsubscribe as

$$\text{unsubscribe}(c,s) \stackrel{\text{def}}{=} \{S_c^A(t+\varepsilon) = S_c^A(t) \backslash s\}, \tag{2.33}$$

where $t$ is a point in time when unsubscribe occurs, $\varepsilon$ is a small positive infinitesimal quantity of time, $S_c^A(t)$ is the set of active subscriptions of $c$ before unsubscribe occurs, and $S_c^A(t+\varepsilon)$ is the set of active subscriptions of $c$ after unsubscribe occurs.

**Definition 2.10** (Notify)**.** Let $c \in \boldsymbol{C}$ be a client, and let $p \in \boldsymbol{P}$ be a publication in a Boolean system. We define an event *notify* as

$$\text{notify}(c,p) \stackrel{\text{def}}{=} \{P_c^R(t+\varepsilon) = P_c^R(t) \cup p\}, \tag{2.34}$$

where $t$ is a point in time when notify occurs, $\varepsilon$ is a small positive infinitesimal quantity of time, $P_c^R(t)$ is the set of publications that $c$ has received before $t$, and $P_c^R(t+\varepsilon)$ is the set of publications which $c$ has received up to $t$ and including $t$.

### 2.2.3 System Variables

In the previous subsection we have defined how events change values of system variables. In this subsection, we define correct values of these variables at different points in time.

**Definition 2.11** (Published Publications)**.** Let $c \in \boldsymbol{C}$ be a client in a Boolean system, let $\tau = (\omega, \zeta)$ be a timed word over an alphabet $\alpha$ of events in this system, and let $t$ be a point in time. We define a finite set $P_c^P(t) \subseteq \boldsymbol{P}$ of publications which $c$ has published before $t$ as follows

$$P_c^P(t) \stackrel{\text{def}}{=} \{p \in \boldsymbol{P} : (\tau, t) \models \widehat{\blacklozenge}\text{publish}(c,p)\}. \tag{2.35}$$

In other words, $P_c^P(t)$ is a subset of the elements of $\boldsymbol{P}$ for which at $t$ there exists a previous event $\text{publish}(c,p)$ in $\omega$.

**Definition 2.12** (Received Publications)**.** Let $c \in \boldsymbol{C}$ be a client in a Boolean system, let $\tau = (\omega, \zeta)$ be a timed word over an alphabet $\alpha$ of events in this system, and let $t$ be a point in time. We define a finite set $P_c^R(t) \subseteq \boldsymbol{P}$ of publications which $c$ has before $t$ as follows

$$P_c^R(t) \stackrel{\text{def}}{=} \{p \in \boldsymbol{P} : (\tau, t) \vDash \widehat{\blacklozenge}\text{notify}(c, p)\}. \tag{2.36}$$

In other words, $P_c^R(t)$ is a subset of the elements of $\boldsymbol{P}$ for which at $t$ there exists a previous event $\text{notify}(c, p)$ in $\omega$.

**Definition 2.13** (Active Subscriptions)**.** Let $c \in \boldsymbol{C}$ be a client in a Boolean system, let $\tau = (\omega, \zeta)$ be a timed word over an alphabet $\alpha$ of events in this system, and let $t$ be a point in time. We define a finite set $S_c^A(t) \subseteq \boldsymbol{S}$ of active subscriptions of $c$ at $t$ as follows

$$S_c^A(t) \stackrel{\text{def}}{=} \{s \in \boldsymbol{S} : (\tau, t) \vDash [\neg\text{unsubscribe}(c, s)]\widehat{\mathcal{S}}\text{subscribe}(c, s)\}. \tag{2.37}$$

In other words, $S_c^A(t)$ is a subset of the elements of $\boldsymbol{S}$ for which at $t$ there exists a previous event $\text{subscribe}(c, s)$ in $\omega$ that is not followed by an event $\text{unsubscribe}(c, s)$ up to $t$.

### 2.2.4 Matching Model

Each subscription in a Boolean system defines an arbitrary Boolean matching function. Since this function does not depend on time, we say that subscriptions are *stateless* filters in the Boolean system. A subscription matching function evaluates the content of a publication for this subscription by returning $\top$ when the publication matches, and $\bot$ when it does not match to a subscription.

**Definition 2.14** (Boolean Matching Function)**.** We define a matching function $m_s : \boldsymbol{P} \mapsto \{\top, \bot\}$ associated to subscription $s \in \boldsymbol{S}$ in a Boolean system as any time-independent Boolean function which assigns $\top$ or $\bot$ to every $p \in \boldsymbol{P}$.

In the rest of this subsection, we use the following simplified notation to denote that a publication $p \in \boldsymbol{P}$ matches or does not match a subscription $s \in \boldsymbol{S}$:

$$p \prec s \stackrel{\text{def}}{=} [m_s(p) = \top] \text{ and} \tag{2.38}$$

$$p \not\prec s \stackrel{\text{def}}{=} [m_s(p) = \bot], \tag{2.39}$$

respectively.

### 2.2.5 Specification

Formally, a *specification* is a set of words which represents the allowed behaviors of a real-time system [2, 144]. A word is valid if a real-time system behaves as allowed. We specify the valid set of words for a Boolean system in MITL. Events *publish*, *subscribe* and *unsubscribe* are independent events produced

by clients in a Boolean system, whereas events *notify* are produced by the system and are causally related to the former events. The allowed behavior of a Boolean system implies the correct notification of subscribed clients in a timely manner, as this is the essence of every publish/subscribe system.

A real-time system is correct with respect to specification $\Sigma$ if it exhibits only the words that are specified in $\Sigma$. Usually, a real-time system specification consists of several different properties, where each is either a safety or liveness property. A *safety* property asserts that "something bad never happens", while a *liveness* property asserts that "something good will eventually happen" in a real-time system [121, 189, 79].

**Definition 2.15** (Boolean Publish/Subscribe System)**.** A Boolean system is a publish/subscribe system with the following specification $\Sigma$:

$$\textit{(Responsiveness)}\ \forall y \in \boldsymbol{C}, \forall s \in \boldsymbol{S} : \Box\{\text{subscribe}(y,s) \rightarrow \widehat{\Diamond}_{\leq \delta_s}[\textit{deliver-matching}(y,s) \mathcal{W} \text{unsubscribe}(y,s)]\}, \tag{2.40}$$

$$\textit{(Validity)}\quad \forall y \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \Box\neg[\text{notify}(y,p) \wedge \nexists x \in \boldsymbol{C} : \widehat{\blacklozenge}_{\leq \delta_p} \text{publish}(x,p)], \tag{2.41}$$

$$\textit{(Legality)}\quad \forall y \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \Box\neg\{\text{notify}(y,p) \wedge \nexists s \in \boldsymbol{S} : [\widehat{\blacklozenge}_{\leq \delta_s} \textit{is-subscribed}(y,s) \wedge (p \prec s)]\}, \text{ and} \tag{2.42}$$

$$\textit{(Uniqueness)}\quad \forall y \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \Box\neg[\text{notify}(y,p) \wedge \widehat{\blacklozenge}\text{notify}(y,p)], \tag{2.43}$$

where we used the following temporal formulas to simplify the presentation:

$$\textit{is-subscribed}(y,s) \stackrel{\text{def}}{=} \{[\neg\text{unsubscribe}(y,s)] \widehat{\mathcal{S}} \text{subscribe}(y,s)\}, \text{ and} \tag{2.44}$$

$$\textit{deliver-matching}(y,s) \stackrel{\text{def}}{=} \forall x \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \{[\text{publish}(x,p) \wedge (p \prec s)] \rightarrow \widehat{\Diamond}_{\leq \delta_p} \text{notify}(y,p)\}, \tag{2.45}$$

and where $\delta_s$ is the *maximal subscription processing delay*, $\delta_p$ is the *maximal publication diffusion delay* in the system, and $\delta_p \leq \delta_s$.

The upper specification defines a liveness property (responsiveness) and three safety properties (legality, validity and uniqueness). The three safety properties forbid delivery of publications which should not be delivered to a subscriber, but do not assume anything about cases when such delivery is required. For example, not delivering any publication in a Boolean system would perfectly satisfy these three properties. On the contrary, the responsiveness property ensures delivery of those publications which have to be delivered, but does not say anything about cases when such delivery is forbidden. For example, delivering each published publication to every subscriber in a Boolean system would perfectly satisfy this property. Therefore, every correctly implemented Boolean publish/subscribe system always satisfies all four properties.

The responsiveness property defined in expression (2.40) states that if a client activates a subscription $s$, then, after a period of time $\leq \delta_s$ and before the cancellation of $s$, every subsequent publishing event producing a matching publication $p \prec s$ will lead to delivery of $p$ to the subscriber during a time interval

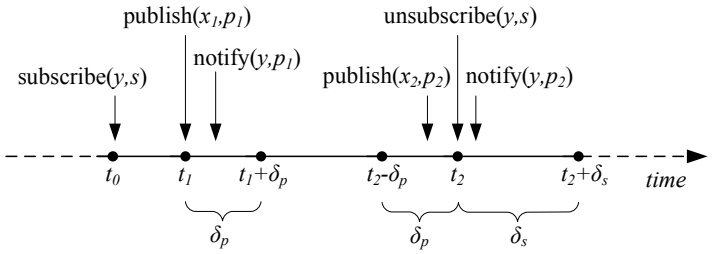



Figure 2.2: Processing an event unsubscribe in a Boolean publish/subscribe system.

$\leq \delta_p$.

According to expression (2.40), every publishing event of a matching publication during a period of time while a client is subscribed will lead to the delivery of this publication to the client in a time $\leq \delta_p$. Please note that this is also true for matching publications published in a time $\leq \delta_p$ before the client unsubscribes.

The *validity* property defined in expression (2.41) states that a publication should never be delivered to a client if it has not been published previously in time $\leq \delta_p$. Therefore, this property forbids delivery of outdated or non-published publications to subscribers.

The *legality* property defined in expression (2.42) states that a publication should not be delivered to a client if it has not been a matching publication to at least one of its subscriptions previously in time $\leq \delta_s$. Please note that this property permits delivery of matching publications in time $\delta s$ after the occurring of an event unsubscribe. In other words, it tolerates that the client receives some matching, but delayed, publications during the processing of events unsubscribe.

**Example 2.1** (Processing of an event unsubscribe)**.** In Figure 2.2 we see a typical situation when a client is subscribed to $s$ during a period of time $(t_0, t_2]$. The publishing event of publication $p_1$ matching subscription $s$ at $t_1$ will lead to the delivery of $p$ to client $y$ in period $(t_1, t_1 + \delta_p]$ in which $y$ is still subscribed. However, the publishing event of publication $p_2$ in period $(t_2 - \delta_p, t_2]$ will also lead to the delivery of $p_2$ in period $(t_2, t_2 + \delta_p]$ because client $y$ has to tolerate delivery of matching publications during the processing of event unsubscribe, i.e. in time $(t_2, t_2 + \delta_s]$.

Finally, the *uniqueness* property defined in expression (2.43) states that a publication should not be delivered to the same client more than once. This is reasonable, because it would not carry any new information for the subscriber.

## 2.3 Top-k/w Publish/Subscribe System

In this section we formally specify the top-k/w publish/subscribe system (*top-k/w system*) using MITL. This system is based on the top-k/w matching model in which each subscription defines an arbitrary, time-independent and real-valued scoring function and two parameters $k \in \mathbb{N}$ and $w \in \mathbb{R}^+$. In this matching model each subscription continuously monitors published publications to identify $k$ best ranked

publications, among publications that are published previously within a time-based window of size $w$, with respect to a scoring function.

### 2.3.1 Structural View

We define a triple $\boldsymbol{B} = (\boldsymbol{C}, \boldsymbol{P}, \boldsymbol{S})$, where $\boldsymbol{C}$ is a finite set of clients, $\boldsymbol{P}$ is a finite set of publications, and $\boldsymbol{S}$ is a finite set of subscriptions in a top-k/w system. $\boldsymbol{B}$ gives the *structural view* of a top-k/w system and determines the boundaries of the system state space. A client $c \in \boldsymbol{C}$ may publish or cancel publications from $\boldsymbol{P}$ and activate or cancel subscriptions from $\boldsymbol{S}$. Therefore, a top-k/w system defines three variables per each client $c \in \boldsymbol{C}$ in the system, one variable per each subscription in the system, and one variable per system:

- a finite set $P_c^P \subseteq \boldsymbol{P}$ of *published publications* (i.e. all publications which a client $c$ has published previously),
- a finite set $P_c^R \subseteq \boldsymbol{P}$ of *received publications* (i.e. all publications which a client $c$ has received previously),
- a finite set $S_c^A \subseteq \boldsymbol{S}$ of *active subscriptions* (i.e. all subscriptions to which a client $c$ is currently subscribed to),
- a finite set $P_s^W \subseteq \boldsymbol{P}$ of *publications within the window* (i.e. all publications which are within the window of a subscription $s$) and
- a finite set $P^A \subseteq \boldsymbol{P}$ of *active publications* (i.e. all publications which have been published and are still not unpublished).

**Definition 2.16** (System Variables)**.** In a top-k/w system, for each client $c_i \in \boldsymbol{C}$ let $V_i \stackrel{\text{def}}{=} \{P_{c_i}^P, P_{c_i}^R, S_{c_i}^A\}$ denote the set of its variables. For each subscription $s_i \in \boldsymbol{S}$, let $P_i^W$ be the set of publications within the window of $s_i$, and let $P^A$ be the set of active publications. Then a finite set $\boldsymbol{V}$ of all variables in this system is defined as

$$\boldsymbol{V} \stackrel{\text{def}}{=} \{V_1, V_2, \ldots, V_{|\boldsymbol{C}|}, P_1^W, P_2^W, \ldots, P_{|\boldsymbol{S}|}^W, P^A\}, \tag{2.46}$$

which has the following cardinality $|\boldsymbol{V}| = 3 \cdot |\boldsymbol{C}| + |\boldsymbol{S}| + 1$.

Analogous to a Boolean system, we assume that all system variables in a top-k/w system are empty sets in the *initial state*.

**Definition 2.17** (Initial State)**.** We define the initial state of a top-k/w system as follows

$$\begin{aligned} &\forall c \in \boldsymbol{C} : P_c^P(t=0) \stackrel{\text{def}}{=} \emptyset, P_c^R(t=0) \stackrel{\text{def}}{=} \emptyset, S_c^A(t=0) \stackrel{\text{def}}{=} \emptyset, \\ &\forall s \in \boldsymbol{S} : P_s^W(t=0) \stackrel{\text{def}}{=} \emptyset, \text{ and} \\ &\qquad P^A(t=0) \stackrel{\text{def}}{=} \emptyset. \end{aligned} \tag{2.47}$$

### 2.3.2 Behavioral View

The following events can occur in a top-k/w system: 1) *publish*, 2) *unpublish*, 3) *subscribe*, 4) *unsubscribe*, 5) *notify* and 6) *drop*. The events *publish*, *subscribe*, *unsubscribe* and *notify* are also present in the Boolean system, while events *unpublish* and *drop* are specific for the top-k/w system. An event *unpublish* denotes the canceling of an already published publication by forbidding its subsequent delivery to subscribers. An event *drop* denotes the dropping of a previously published publication from a subscription window. The corresponding event *publish* precedes every *drop* by the size of the subscription window. These six events determine the alphabet of events in a top-k/w system, where each possible event in the system is represented by a single letter of this alphabet.

**Definition 2.18** (Alphabet)**.** Let $\boldsymbol{C}$ be a finite set of clients, let $\boldsymbol{P}$ be a finite set of publications, and let $\boldsymbol{S}$ be a finite set of subscriptions in a top-k/w system. An alphabet $\alpha$ of events in this system consists of the following letters:

$$\begin{aligned}
&\forall c \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \text{publish}(c,p) \in \alpha,\\
&\forall c \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \text{unpublish}(c,p) \in \alpha,\\
&\forall c \in \boldsymbol{C}, \forall s \in \boldsymbol{S} : \text{subscribe}(c,s) \in \alpha,\\
&\forall c \in \boldsymbol{C}, \forall s \in \boldsymbol{S} : \text{unsubscribe}(c,s) \in \alpha,\\
&\forall c \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \text{notify}(c,p) \in \alpha, \text{ and}\\
&\forall s \in \boldsymbol{S}, \forall p \in \boldsymbol{P} : \text{drop}(s,p) \in \alpha,
\end{aligned} \tag{2.48}$$

and has the following cardinality $|\alpha| = |\boldsymbol{C}||\boldsymbol{P}| + |\boldsymbol{C}||\boldsymbol{P}| + |\boldsymbol{C}||\boldsymbol{S}| + |\boldsymbol{C}||\boldsymbol{S}| + |\boldsymbol{C}||\boldsymbol{P}| + |\boldsymbol{S}||\boldsymbol{P}| = |\boldsymbol{C}|(3 \cdot |\boldsymbol{P}| + 2 \cdot |\boldsymbol{S}|) + |\boldsymbol{S}||\boldsymbol{P}|$.

In the rest of this subsection we define changes that an occurring event causes to system variables of a top-k/w system. Analogous to a Boolean system, system variables of a top-k/w system do not change their values between each two consecutive events and thus the system remains in the same state during this interval as shown in Figure 2.1.

**Definition 2.19** (Publish)**.** Let $c_i \in \boldsymbol{C}$ be a client, and let $p \in \boldsymbol{P}$ be a publication in a top-k/w system. We define an event publish as

$$\begin{aligned}
\text{publish}(c_i,p) \stackrel{\text{def}}{=} \{&P^P_{c_i}(t+\varepsilon) = P^P_{c_i}(t) \cup p,\\
&P^A(t+\varepsilon) = P^A(t) \cup p, \text{ and}\\
\forall c_j \in \boldsymbol{C}, \forall s \in S^A_{c_j} : &P^W_s(t+\varepsilon) = P^W_s(t) \cup p\},
\end{aligned} \tag{2.49}$$

where $t$ is a time point when publish occurs, $\varepsilon$ is a small positive infinitesimal quantity of time, $P^P_{c_i}(t)$ is the set of publications that $c_i$ has published before $t$, $P^P_c(t+\varepsilon)$ is the set of publications which $c$ has published up to $t$ and including $t$, $P^A(t+\varepsilon)$ is the set of active publications before publish occurs, $P^A(t)$ is the set of active publications after publish occurs, $P^W_s(t+\varepsilon)$ is the set of publications within the window

of a subscription $s$ before publish occurs, and $P_s^W(t)$ is the set of publications within the window of $s$ after publish occurs.

**Definition 2.20** (Unpublish)**.** Let $c \in \boldsymbol{C}$ be a client, and let $p \in \boldsymbol{P}$ be a publication in a top-k/w system. We define an event unpublish as

$$\text{unpublish}(c,p) \stackrel{\text{def}}{=} \{P^A(t+\varepsilon) = P^A(t) \setminus p\}, \tag{2.50}$$

where $t$ is a point in time when unpublish occurs, $\varepsilon$ is a small positive infinitesimal quantity of time, $P^A(t)$ is the set of active publications before unpublish occurs, and $P^A(t+\varepsilon)$ is the set of active publications after unpublish occurs.

**Definition 2.21** (Subscribe)**.** Let $c \in \boldsymbol{C}$ be a client, and let $s \in \boldsymbol{S}$ be a subscription in a top-k/w system. We define an event subscribe as

$$\text{subscribe}(c,s) \stackrel{\text{def}}{=} \{S_c^A(t+\varepsilon) = S_c^A(t) \cup s\}, \tag{2.51}$$

where $t$ is a point in time when subscribe occurs, $\varepsilon$ is a small positive infinitesimal quantity of time, $S_c^A(t)$ is the set of active subscriptions of $c$ before subscribe occurs, and $S_c^A(t+\varepsilon)$ is the set of active subscriptions of $c$ after subscribe occurs.

**Definition 2.22** (Unsubscribe)**.** Let $c \in \boldsymbol{C}$ be a client, and let $s \in \boldsymbol{S}$ be a subscription in a top-k/w system. We define an event unsubscribe as

$$\begin{aligned}\text{unsubscribe}(c,s) \stackrel{\text{def}}{=} \{&S_c^A(t+\varepsilon) = S_c^A(t) \setminus s \text{ and} \\ &P_s^W(t+\varepsilon) = \emptyset\},\end{aligned} \tag{2.52}$$

where $t$ is a point in time when unsubscribe occurs, $\varepsilon$ is a small positive infinitesimal quantity of time, $S_c^A(t)$ is the set of active subscriptions of $c$ before unsubscribe occurs, $S_c^A(t+\varepsilon)$ is the set of active subscriptions of $c$ after unsubscribe occurs, and $P_s^W(t+\varepsilon)$ is the set of publications within the window of $s$ after unsubscribe occurs.

**Definition 2.23** (Notify)**.** Let $c \in \boldsymbol{C}$ be a client, and let $p \in \boldsymbol{P}$ be a publication in a top-k/w system. We define an event *notify* as

$$\text{notify}(c,p) \stackrel{\text{def}}{=} \{P_c^R(t+\varepsilon) = P_c^R(t) \cup p\}, \tag{2.53}$$

where $t$ is a point in time when notify occurs, $\varepsilon$ is a small positive infinitesimal quantity of time, $P_c^R(t)$ is the set of publications that $c$ has received before $t$, and $P_c^R(t+\varepsilon)$ is the set of publications which $c$ has received up to $t$ and including $t$.

**Definition 2.24** (Drop)**.** Let $s \in S_c^A \subseteq \boldsymbol{S}$ be an active subscription of a client $c \in \boldsymbol{C}$, and let $p \in \boldsymbol{P}$ be a publication in a top-k/w system. We define an event drop as

$$\text{drop}(s,p) \stackrel{\text{def}}{=} \{P_s^W(t+\varepsilon) = P_s^W(t) \setminus p\}, \tag{2.54}$$

where $t$ is a point in time when $p$ is dropped from the window of $s$, $\varepsilon$ is a small positive infinitesimal quantity of time, $P_s^W(t)$ is the set of publications within the window of $s$ before drop occurs, and $P_s^W(t+\varepsilon)$ is the set of publications within the window of $s$ after the drop occurs.

### 2.3.3 System Variables

In the previous subsection we have defined how each event affects the values of top-k/w system variables. In this subsection, we define the correct values of these variables at different points in time.

**Definition 2.25** (Published Publications)**.** Let $c \in \boldsymbol{C}$ be a client in a top-k/w system, let $\tau = (\omega, \zeta)$ be a timed word over an alphabet $\alpha$ of events in this system, and let $t$ be a point in time. We define a finite set $P_c^P(t) \subseteq \boldsymbol{P}$ of publications which $c$ has published before $t$ as

$$P_c^P(t) \stackrel{\text{def}}{=} \{p \in \boldsymbol{P} : (\tau, t) \vDash \widehat{\blacklozenge}\text{publish}(c,p)\}. \tag{2.55}$$

In other words, $P_c^P(t)$ is a subset of the elements of $\boldsymbol{P}$ for which at $t$ there exists a previous event $\text{publish}(c,p)$ in $\omega$.

**Definition 2.26** (Received Publications)**.** Let $c \in \boldsymbol{C}$ be a client in a top-k/w system, let $\tau = (\omega, \zeta)$ be a timed word over an alphabet $\alpha$ of events in this system, and let $t$ be a point in time. We define a finite set $P_c^R(t) \subseteq \boldsymbol{P}$ of publications which $c$ has received before $t$ as

$$P_c^R(t) \stackrel{\text{def}}{=} \{p \in \boldsymbol{P} : (\tau, t) \vDash \widehat{\blacklozenge}\text{notify}(c,p)\}. \tag{2.56}$$

In other words, $P_c^P(t)$ is a subset of the elements of $\boldsymbol{P}$ for which at $t$ there exists a previous event $\text{notify}(c,p)$ in $\omega$.

**Definition 2.27** (Active Subscriptions)**.** Let $c \in \boldsymbol{C}$ be a client in a top-k/w system, let $\tau = (\omega, \zeta)$ be a timed word over an alphabet $\alpha$ of events in this system, and let $t$ be a point in time. We define a finite set $S_c^A(t) \subseteq \boldsymbol{S}$ of active subscriptions of $c$ in $t$ as

$$S_c^A(t) \stackrel{\text{def}}{=} \{s \in \boldsymbol{S} : (\tau, t) \vDash [\neg\text{unsubscribe}(c,s)]\widehat{\mathcal{S}}\text{subscribe}(c,s)\}. \tag{2.57}$$

In other words, $S_c^A(t)$ is a subset of the elements of $\boldsymbol{S}$ for which at $t$ there exists a previous event $\text{subscribe}(c,s)$ in $\omega$ that not followed by an event $\text{unsubscribe}(c,s)$ up to $t$.

**Definition 2.28** (Active Publications)**.** Let $\tau = (\omega, \zeta)$ be a timed word over an alphabet $\alpha$ of a top-k/w system, and let $t$ be a point in time. We define a finite set $P^A \subseteq \boldsymbol{S}$ of active publications in a top-k/w system at $t$ as follows

$$P^A(t) \stackrel{\text{def}}{=} \{p \in \boldsymbol{P} : (\tau, t) \vDash [\exists c \in \boldsymbol{C} : (\neg\text{unpublish}(c,p))\widehat{\mathcal{S}}\text{publish}(c,p)]\}. \tag{2.58}$$

In other words, $P^A(t)$ is a subset of the elements of $\boldsymbol{P}$ for which at $t$ there exists a previous event $\text{publish}(c,p)$ in $\omega$ that is not followed by an event $\text{unpublish}(c,p)$ up to $t$.

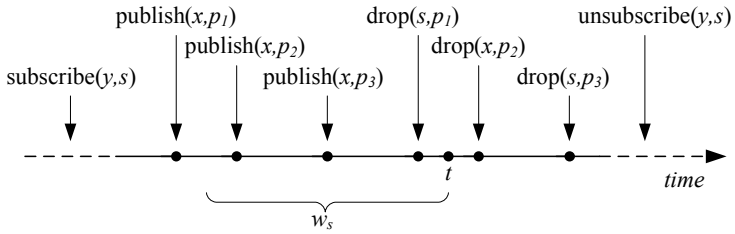



Figure 2.3: Publications within a top-k/w subscription window.

**Definition 2.29** (Publications within a Subscription Window)**.** Let $s \in \boldsymbol{S}$ be a subscription of a client $y \in \boldsymbol{C}$ in a top-k/w system, let $w_s$ be the size of the window of $s$, let $\tau = (\omega, \zeta)$ be a timed word over an alphabet $\alpha$ of events in this system, and let $t$ be a point in time. We define a finite set $P_s^W(t) \subseteq \boldsymbol{P}$ of publications that are within the window of $s$ at $t$ as

$$P_s^W(t) \stackrel{\text{def}}{=} \{p \in \boldsymbol{P} : (\tau, t) \vDash [\exists x \in \boldsymbol{C} : (\textit{is-subscribed}(y,s) \wedge \neg \text{drop}(s,p)) \widehat{\mathcal{S}} \text{publish}(x,p)]\}, \tag{2.59}$$

where $\textit{is-subscribed}(y,s) \stackrel{\text{def}}{=} \{[\neg \text{unsubscribe}(y,s)] \widehat{\mathcal{S}} \text{subscribe}(y,s)\}$.

In other words, $P_s^W(t)$ is a subset of the elements of $\boldsymbol{P}$ for which at $t$ there exists a previous event $\text{publish}(c,p)$ in $\omega$ which was preceded by event $\text{subscribe}(y,s)$, and up to $t$ is neither followed by event $\text{unsubscribe}(y,s)$ nor event $\text{drop}(s,p)$. We assume that top-k/w subscriptions reference future publications, i.e. publications published after subscription activation. In other words, these subscriptions cannot reference publications that are published before subscription activation, but still not expired at the time of subscription activation.

For each subscription $s \in \boldsymbol{S}$, the size $w_s$ of its window is constant in time. As a consequence, for every subscription which is 1) active when an event $\text{publish}(c,p)$ occurs and 2) continues to be active afterwards, the system will produce a corresponding event $\text{drop}(s,p)$ exactly $w_s$ time units after the publish event. Since different subscriptions have different sizes of their windows, a previously published publication will be dropped from their windows at different points in time. Therefore, we can write expression (2.59) equivalently as follows

$$P_s^W(t) \stackrel{\text{def}}{=} \{p \in \boldsymbol{P} : (\tau, t) \vDash [\exists x \in \boldsymbol{C} : \textit{is-subscribed}(y,s) \widehat{\mathcal{S}}_{\leq w_s} \text{publish}(x,p)]\}, \tag{2.60}$$

where $w_s$ is the size of the window of subscription $s$.

**Example 2.2** (Publications within a subscription window)**.** In Figure 2.3 we see that publication $p_2$ and $p_3$ are within the window of $s$ at $t$, while publication $p_1$ is not within the window of $s$ at $t$ because it has been dropped from the window of $s$ before $t$.

**Example 2.3** (Dropping publications from subscription windows)**.** In Figure 2.4 we see that a publication $p$ which is published by client $c_4$ at a point in time $t$ will be dropped from the windows of active subscriptions $s_1$, $s_2$ and $s_3$ at points in time $t + w_{s1}$, $t + w_{s2}$ and $t + w_{s3}$, respectively.

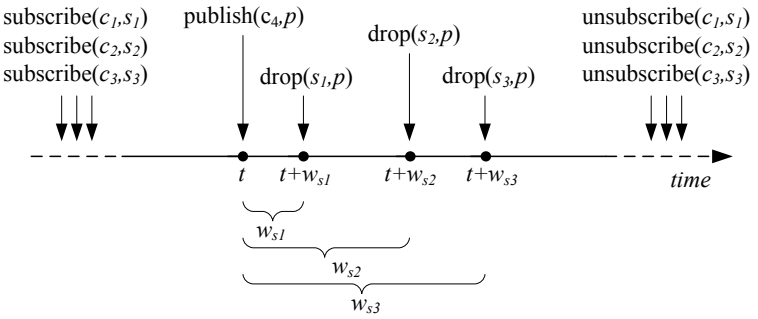



Figure 2.4: Dropping publications from top-k/w subscription windows.

### 2.3.4 Matching Model

In this subsection we present the top-k/w publish/subscribe matching model (*top-k/w matching model*). In this matching model, each subscription $s \in \boldsymbol{S}$ defines an arbitrary, time-independent and real-valued *scoring function* and two parameters $k \in \mathbb{N}$ and $w \in \mathbb{R}^+$. A subscription scoring function evaluates the content of publications for this subscription by returning different scores for different publications. The scores of publications given by a scoring function do not change in time. The three most popular categories of scoring functions are *aggregation*, *distance* and *relevance* functions. The former two categories are used for structured data, while the latter category is used for unstructured data (i.e. text). All these three categories of scoring functions are supported by the top-k/w matching model. In practice, a scoring function may be defined per each subscription in a top-k/w system or once per top-k/w system.

**Definition 2.30** (Scoring Function)**.** We define a scoring function $u_s : \boldsymbol{P} \mapsto \mathbb{R}$ of a subscription $s \in \boldsymbol{S}$ in a top-k/w system as any real-valued function which assigns a score to every $p \in \boldsymbol{P}$, and where $bs(u_s) \in \mathbb{R}$ and $ws(u_s) \in \mathbb{R}$ are the best and worst score it can assign, respectively.

To enable publication ordering by either decreasing or increasing scores, we define a score comparator which we use for the comparison of publication scores, i.e. for ranking publications within the window of a subscription. Notice that, since publication content is static, i.e. it does not change in time, the scores are also static, but the corresponding ranks may change in time with new publish and drop events.

**Definition 2.31** (Score Comparator)**.** Let $u_s$ be a scoring function of a subscription $s \in \boldsymbol{S}$ in the top-k/w system, and let $p, p' \in \boldsymbol{P}$. We define the score comparator $\overset{s}{\rhd}: \boldsymbol{P} \times \boldsymbol{P} \mapsto \{\top, \bot\}$ of $s$ as the following Boolean function

$$p \overset{s}{\rhd} p' \overset{\text{def}}{=} \begin{cases} \top & \text{if and only if } u_s(p) \text{ assigns a higher rank to } p \text{ than } u_s(p') \text{ to } p', \\ \bot & \text{otherwise.} \end{cases} \tag{2.61}$$

The score comparator compares every two publications by their scores such that the result is $\top$ when a higher rank is assigned to the first argument ($p$) than to the second argument ($p'$), or $\bot$ otherwise. For example, $p_1 \overset{s}{\rhd} p_2 \overset{s}{\rhd} \ldots p_k$ denotes a ranked list of $k$ publications for a subscription $s$. Obviously, $p$ and $p'$ are non-commutative for $\overset{s}{\rhd}$, and thus we define an inverse score comparator $\overset{s}{\lhd}: \boldsymbol{P} \times \boldsymbol{P} \mapsto \{\top, \bot\}$ as

follows $p \overset{s}{\lhd} p' \overset{\text{def}}{=} \neg(p \overset{s}{\rhd} p')$.

A publication matches a subscription at a point in time if it is one of top-k publications within the current subscription window. As each subscription window is sliding in time, the result of a matching function is time dependent and thus subscriptions are *statefull* filters in the top-k/w matching model.

**Definition 2.32** (Top-k/w Matching Function)**.** Let $s \in \boldsymbol{S}$ be a subscription in a top-k/w system with parameters $k_s \in \mathbb{N}$ and $w_s \in \mathbb{R}^+$, let $u_s$ and $\overset{s}{\rhd}$ be a scoring function of $s$ and its associated score comparator, and let $ws(u_s) \in \mathbb{R}$ be the worst score that $u_s$ can assign. We define the matching function $m_s : \boldsymbol{P} \mapsto \{\top, \bot\}$ of $s$ as follows

$$\begin{aligned}\{(\tau,t) \vDash m_s(p)\} \overset{\text{def}}{=} [u_s(p) \neq ws(u_s)] \wedge \nexists p_1, p_2, \ldots, p_{k_s} \in \boldsymbol{P} : \{&[(p_1 \in P_s^W(t)) \wedge (p_1 \overset{s}{\rhd} p)] \wedge \\ &[(p_2 \in P_s^W(t)) \wedge (p_2 \overset{s}{\rhd} p)] \wedge \\ &\ldots \\ &[(p_{k_s} \in P_s^W(t)) \wedge (p_{k_s} \overset{s}{\rhd} p)]\} \end{aligned} \tag{2.62}$$

In other words, a publication $p$ will match a subscription $s$ if it neither has the worst score for $u_s$, nor there are $k_s$ or more publications better scored than $p$ within the current window of $s$. From expressions (2.59) and (2.60) we know that publications within the window of a subscription $s$ at a point in time $t$ are those which are previously published and are still not dropped from the window of $s$ at $t$, i.e. that are published in the interval $[t - w_s, t)$.

Expression (2.62) uses system variables, but it can be written without these variables using expressions (2.59) and (2.60):

$$\begin{aligned}\{(\tau,t) \vDash m_s(p)\} \overset{\text{def}}{=} [u_s(p) \neq ws(u_s)] \wedge& \\ \nexists p_1, p_2, \ldots, p_{k_s} \in \boldsymbol{P} : \{&[(\exists x_1 \in \boldsymbol{C} : \textit{is-subscribed}(y,s)\widehat{\mathcal{S}}_{\leq w_s}\text{publish}(x_1,p_1)) \wedge (p_1 \overset{s}{\rhd} p)] \wedge \\ &[(\exists x_2 \in \boldsymbol{C} : \textit{is-subscribed}(y,s)\widehat{\mathcal{S}}_{\leq w_s}\text{publish}(x_2,p_2)) \wedge (p_2 \overset{s}{\rhd} p)] \wedge \\ &\ldots \\ &[(\exists x_{k_s} \in \boldsymbol{C} : \textit{is-subscribed}(y,s)\widehat{\mathcal{S}}_{\leq w_s}\text{publish}(x_{k_s},p_{k_s})) \wedge (p_{k_s} \overset{s}{\rhd} p)]\}. \end{aligned} \tag{2.63}$$

In the rest of this subsection, and analogous to the Boolean system, we use the following simplified notation to denote that publication $p \in \boldsymbol{P}$ matches or does not match subscription $s \in \boldsymbol{S}$:

$$p \prec s \overset{\text{def}}{=} [m_s(p) = \top], \text{ and} \tag{2.64}$$

$$p \not\prec s \overset{\text{def}}{=} [m_s(p) = \bot], \tag{2.65}$$

respectively.

However, please note that the function $m_s(p)$ defined in expressions (2.63) and (2.62) is a function of time, while Definition 2.14 for the Boolean matching function is not. Each variable $P_s^W$ of every active $s \in \boldsymbol{S}$ changes after each new publishing and dropping event according to expressions (2.49) and

(2.54). Therefore, in every implementation of the top-k/w system, we need to check if these variables have changed due to the occurrence of a new event publish or drop.

### 2.3.5 Specification

Events *publish*, *unpublish*, *subscribe* and *unsubscribe* are independent events produced by clients in a top-k/w system, whereas events *drop* and *notify* are produced by the system and are causally related to the former events. Analogous to the Boolean system, the correct behavior of a top-k/w system implies timely notification of subscribed clients, as this is the essence of every publish/subscribe system.

**Definition 2.33** (Top-k/w System)**.** A top-k/w system is a publish/subscribe system with the following specification $\Sigma$:

$$\begin{aligned}
&\textit{(Punctuality)} && \forall x,y \in \boldsymbol{C}, \forall p \in \boldsymbol{P}, \forall s \in \boldsymbol{S} : \Box\{[\text{publish}(x,p) \wedge \textit{is-subscribed}(y,s)] \rightarrow \\
&&& \quad [\textit{exact-drop}(s,p) \vee \widehat{\Diamond}_{\leq w_s}\text{unsubscribe}(y,s)]\}, && (2.66)\\
&\textit{(Responsiveness)} && \forall y \in \boldsymbol{C}, \forall s \in \boldsymbol{S} : \Box\{\text{subscribe}(y,s) \rightarrow \\
&&& \quad \widehat{\Diamond}_{\leq \delta_s}[\textit{deliver-matching}(y,s)\mathcal{W}\text{unsubscribe}(y,s)]\}, && (2.67)\\
&\textit{(Validity)} && \forall y \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \Box\neg[\text{notify}(y,p) \wedge \neg\blacklozenge_{\leq \delta_p}\textit{is-active}(p)], && (2.68)\\
&\textit{(Legality)} && \forall y \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \Box\neg\{\text{notify}(y,p) \wedge \\
&&& \quad \nexists s \in \boldsymbol{S} : \blacklozenge_{\leq \delta_s}[\textit{in-window}(s,p) \wedge (p \prec s)]\}, \text{ and} && (2.69)\\
&\textit{(Uniqueness)} && \forall y \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \Box\neg[\text{notify}(y,p) \wedge \widehat{\blacklozenge}\text{notify}(y,p)], && (2.70)
\end{aligned}$$

where we used the following temporal formulas to simplify the presentation:

$$\begin{aligned}
\textit{is-active}(p) &\stackrel{\text{def}}{=} \exists x \in \boldsymbol{C} : \{[\neg\text{unpublish}(x,p)]\widehat{\mathcal{S}}\text{publish}(x,p)\}, && (2.71)\\
\textit{exact-drop}(s,p) &\stackrel{\text{def}}{=} [\neg\text{drop}(s,p)]\widehat{\mathcal{U}}_{=w_s}\text{drop}(s,p), && (2.72)\\
\textit{in-window}(y,s,p) &\stackrel{\text{def}}{=} \exists x \in \boldsymbol{C} : \textit{is-subscribed}(y,s)\widehat{\mathcal{S}}_{\leq w_s}\text{publish}(x,p), && (2.73)\\
\textit{is-subscribed}(y,s) &\stackrel{\text{def}}{=} \{[\neg\text{unsubscribe}(y,s)]\widehat{\mathcal{S}}\text{subscribe}(y,s)\}, \text{ and} && (2.74)\\
\textit{deliver-matching}(y,s) &\stackrel{\text{def}}{=} \forall x \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \{[(p \prec s) \wedge \textit{in-window}(s,p) \wedge \textit{is-active}(p) \wedge \\
&\qquad \neg\widehat{\blacklozenge}\text{notify}(y,p)] \rightarrow \Diamond_{\leq \delta_p}\text{notify}(y,p)\}. && (2.75)
\end{aligned}$$

The previous specification consists of the punctuality property, a liveness property (responsiveness) and three safety properties (legality, validity and uniqueness). Punctuality is related to dropping of publications from subscription windows, while other properties are related to the delivery of publications to subscribers.

*Punctuality* states that while a subscription is active, each new publishing event will be followed by the corresponding drop or unsubscribe event after exactly $w_s$ time units. The form of this property is such that it combines a safety and liveness property. Using expression (2.27), we can divide it into the

following two properties:

$$\textit{(Regularity)} \quad \forall x, y \in \boldsymbol{C}, \forall p \in \boldsymbol{P}, \forall s \in \boldsymbol{S} : \Box\{[\text{publish}(x,p) \wedge \textit{is-subscribed}(y,s)] \rightarrow \widehat{\Diamond}_{\leq w_s}[\text{drop}(s,p) \vee \text{unsubscribe}(y,s)]\}, \text{ and} \tag{2.76}$$

$$\textit{(Correctness)} \; \forall c \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \Box\neg[\text{publish}(c,p) \wedge \exists s \in \boldsymbol{S} : \widehat{\Diamond}_{<w_s}\text{drop}(s,p)], \tag{2.77}$$

where regularity is a liveness property, and correctness is a safety property. *Regularity* states that while a subscription is active, each new publishing event will be followed by the corresponding drop or unsubscribe event in a time $\leq w_s$, while *correctness* states that the time period between a publication's publishing and its dropping from a subscription window should never be shorter than the size $w_s$ of the subscription window. Therefore, regularity requires that publication droppings happen in certain cases, but does not say anything about the situations when such droppings should not happen. For example, immediate dropping of each published publication would perfectly satisfy this property. On the contrary, correctness forbids dropping a publication from a subscription window in time period $w_s$ after the publishing, but does not imply anything about cases when such dropping should happen. For example, a system which never drops any publication would perfectly satisfy this property.

The *responsiveness* property defined in expression (2.67) states that if a client activates a subscription, then, after at most a time period $\delta_s$, and before subscription cancellation, every subsequent publishing of a publication which is at a point in time $t$ 1) active, 2) within the subscription window, 3) if it matches the subscription and 4) has not already been delivered to the client, will lead to publication delivery to the subscriber within time period $\leq \delta_p$.

The *validity* property defined in expression (2.68) states that a publication should never be delivered to a client if it has not been active previously in time $\leq \delta_p$. Therefore, this property forbids delivery of outdated or non-published publications to subscribers.

The *legality* property defined in expression (2.69) states that a publication should never be delivered to a client if it has not been a top-k publication within the window of one of its subscriptions previously in time $\leq \delta_s$. Due to the processing of unsubscribe events, and analogous to the Boolean system, each client in a top-k/w system has to tolerate the delivery of matching, but delayed, publications during at most a time period $\delta_s$ after subscription cancellation, as shown in Figure 2.2. Additionally, this property forbids delivery of invalid publications, i.e. those which have been published, but are dropped from a subscription window.

Finally, the *uniqueness* property defined in expression (2.70) states that a publication should never be delivered to the same client more than once, which is reasonable because it would not carry any new information for the subscriber.

## 2.4 Discussion

In this subsection we first compare specifications of the Boolean and top-k/w systems. After that, we discuss how different values of subscription parameters $k$ and $w$ influence the top-k/w matching model.

At the end of this section, we prove that the every Boolean system is a special case of the corresponding top-k/w system.

### 2.4.1 Comparison of the Boolean and Top-k/w Publish/Subscribe Systems

The structural view of both the Boolean and top-k/w system defines identical triples $\boldsymbol{B} = (\boldsymbol{C}, \boldsymbol{P}, \boldsymbol{S})$, where $\boldsymbol{C}$ is a finite set of clients, $\boldsymbol{P}$ is a finite set of publications, and $\boldsymbol{S}$ is a finite set of subscriptions in a system. Top-k/w subscriptions are different from Boolean subscriptions because a top-k/w subscription $s$ is defined as a real-valued scoring function $u_s$ associated a score comparator $\overset{s}{\rhd}$ and parameters $k_s \in \mathbb{N}$ and $w_s \in \mathbb{R}^+$, while a Boolean subscription $s$ defines just a Boolean matching function $m_s$. Besides system variables $P_c^P \subseteq \boldsymbol{P}, P_c^R \subseteq \boldsymbol{P}$, and $S_c^A \subseteq \boldsymbol{S}$, which are in both systems defined for each client $c \in \boldsymbol{C}$, the top-k/w system additionally defines a variable $P_s^W \subseteq \boldsymbol{P}$ for each subscription $s \in \boldsymbol{S}$, and variable $P^A \subseteq \boldsymbol{P}$ for the whole system.

In the behavioral view, events *publish*, *subscribe*, *unsubscribe* and *notify* are defined in both systems. The top-k/w system additionally defines events *unpublish* and *drop*. In the Boolean system, a matching publication is always delivered in time $\leq \delta_p$ after publication, where $\delta_p$ is usually very small. On the contrary, in the top-k/w system, the delivery of a published publication may be delayed for a period of time which is less or equal to the size of a subscription window $w_s$. Therefore, as subscription windows can be large, it is reasonable to support the event *unpublish* in the top-k/w system to invalidate a previously published and postponed publication. Additionally, the definitions of the event *publish* (Definition 2.7 and Definition 2.19) are different because in the top-k/w system, every newly published publication is not only added to variable $P_c^P$ of its publisher $c \in \boldsymbol{C}$, but also to variable $P^A$ of the top-k/w system and to variable $P_s^W$ of each active subscription $s \in \boldsymbol{S}$. Similarly, the definitions of the event *unsubscribe* (Definition 2.9 and Definition 2.22) are also different since the cancelation of a top-k/w subscription clears its variable $P_s^W$.

The main difference between the Boolean and top-k/w system is related to the process of matching publications to subscriptions. Actually, all of the previously identified differences between these two systems are directly related to different matching models. In the Boolean system, subscriptions are stateless filters such that a matching result is time-independent and relies only on the matching function. On the contrary, subscriptions in the top-k/w system are statefull filters, where a matching result depends on the comparison of a publication score calculated by a time-independent scoring function and scores of other publications belonging to a current subscription window.

Hereafter we discuss the differences in specifications of the Boolean and top-k/w systems.

- The punctuality property defined in expression (2.66) is related to dropping publications from subscription windows, and is specific for the top-k/w system. Besides that, both specifications define responsiveness, validity, legality and uniqueness properties.

- Since multiple deliveries of a publication to the same client are forbidden in both systems, the related uniqueness properties defined in expressions (2.43) and (2.70) are identical.

- The validity properties defined in expressions (2.41) and (2.68) are very similar, since the only difference is related to the additional event *unpublish* that occurs in top-k/w systems. Due to this event, the validity property of the top-k/w system defines an additional restriction on a publication delivery, requiring that a publication has to be active (i.e. published and not yet canceled) previously in time $\leq \delta_p$.

- The legality properties, which are defined in expressions (2.42) and (2.69), are also very similar. The only difference is related to the additional requirement in the top-k/w system which states that every publication delivered to a subscriber has to be within a subscription window previously in a time $\leq \delta_s$. Please recall that matching functions are time-independent in the Boolean system, and time-dependent in the top-k/w system.

- The responsiveness properties defined in expressions (2.40) and (2.67) look identical, but are actually quite different due the differently defined delivery conditions *deliver-matching*$(y,s)$ in expressions (2.45) and (2.75). In the Boolean system, a matching publication is always delivered to a subscriber in time $\leq \delta_p$ after its publication: $[\text{publish}(x,p) \wedge (p \prec s)] \rightarrow \widehat{\Diamond}_{\leq \delta_p}\text{notify}(y,p)$. On the contrary, in the top-k/w system, a publication may become matching for a subscription at any point in time between its publishing and dropping from the subscription window. Every publication which is at a point in time within this interval 1) active, 2) within the subscription window, 3) matching to the subscription and 4) has not already been delivered to the client, will be delivered to the subscriber in time $\leq \delta_p$: $[(p \prec s) \wedge \textit{in-window}(s,p) \wedge \textit{is-active}(p) \wedge \neg\widehat{\blacklozenge}\text{notify}(y,p)] \rightarrow \Diamond_{\leq \delta_p}\text{notify}(y,p)$.

### 2.4.2 Subscription Parameters in the Top-k/w Matching Model

In this subsection we discuss a top-k/w subscription under different values of its subscription parameters $k_s \in \mathbb{N}$ and $w_s \in \mathbb{R}^+$. We will assume that $t$ is a point in time when a publication $p \in \boldsymbol{P}$ is published and when $s$ is active. Besides the usual case with the typical values of subscription parameters, four extreme cases of these parameters are possible. The first three cases are related to extreme values of subscription parameters, while the fourth is the combination of values of these parameters and the intensity of publishing in a top-k/w system.

**Parameter $\mathbf{k_s} \rightarrow \infty$.** In this case, each published publication $p$ with a score better than $ws(u_s)$ will match $s$ since an infinite number of publications with better scores than $p$ within the window of $s$ at $t$ does not exist according to expressions (2.62) and (2.63). As a consequence every published publication whose score is better than $ws(u_s)$ will be delivered to client $c$ in time $\leq \delta_p$ as stated in expression (2.40). We refer to this extreme case as the top-$\infty$.

**Parameter $\mathbf{w_s} \rightarrow 0$.** This case is very similar to the previous one as each $p$ with a better score than $ws(u_s)$ will be the only publication within the window of $s$ at $t$, and thus will match $s$ according to expressions (2.62) and (2.63). As a consequence, every published publication whose score is better

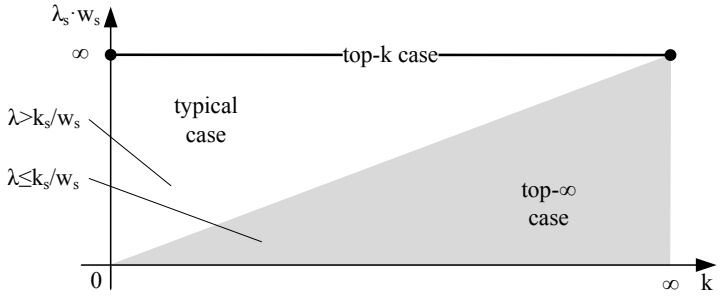



Figure 2.5: Special cases of subscription parameters in the top-k/w system.

than $ws(u_s)$ will be delivered to client $c$ in time $\leq \delta_p$ as stated in expression (2.67). Therefore, this extreme case is also the top-$\infty$ case.

**Parameter $\mathbf{w_s} \to \infty$.** In this case, the window of $s$ does not slide, but expands in time. Therefore, only publications that are among $k_s$ best scored publications published from the activation of $s$ will be delivered to $c$. As time passes it would be progressively harder and harder for newly published publications to become top-k publications and to be delivered to $c$.

**Low intensity of publishing.** Let us model the publishing process in the top-k/w system as a homogeneous Poisson process with parameters $\lambda \leq k_s/w_s$ and $\tau = w_s$, where $\lambda$ is the expected number of publish events that occur in time $\tau$. In this case, a number of newly published publications in every interval $w_s$ is smaller than $k_s$ and thus, according to expressions (2.62) and (2.63), all publications with scores better than $ws(u_s)$ will be delivered to client $c$. Therefore, this extreme case is also the top-$\infty$ case.

**Typical case.** The last combination of subscription parameters excludes the four extreme cases mentioned above. In this case, subscription parameters have typical non-extreme values, while the intensity of publishing is related to subscription parameters as follows: $\lambda > k_s/w_s$. Therefore, this subscription acts as a filter for published publications since the number of matching publications will be larger than in the top-k case, but smaller than in the top-$\infty$ case. In Figure 2.5, we see a graphical representation of the typical, top-k and top-$\infty$ cases. This figure graphically shows that the typical case is located between the top-k and top-$\infty$ cases.

**Example 2.4** (An infinite subscription window)**.** In Figure 2.6 we see an example of top-k/w subscription for $k_s = 1, w_s \to \infty$ and time-independent publication scores. This figure shows the number of published publications which will probably be required for delivering the $n$-th matching publication, i.e. $n$-th top-1 publication since subscription activation. In this example we have used the odds theorem [35, 36] to calculate the required probabilities. We see that after 1 billion publications we will have just 22 publications which are top-1 within the subscription window.

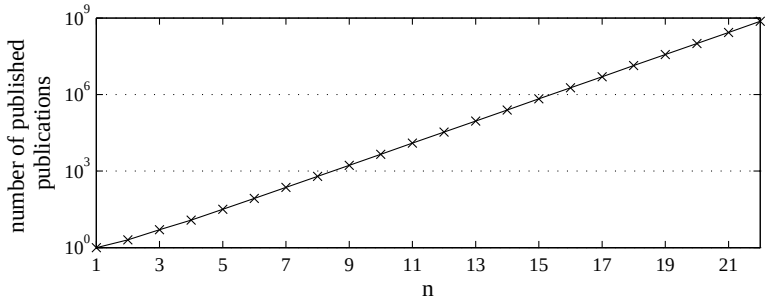



Figure 2.6: Number of matching publications for a top-k/$\infty$ subscription and time-independent publication scores.

### 2.4.3 Relationship Between the Boolean and Top-k/w System

In this subsection we show that the top-k/w system is more general than the Boolean system, i.e. that every Boolean system is a special case of the corresponding top-k/w system. We can easily achieve that a top-k/w system behaves as a Boolean system by selecting proper values of subscription parameters and scoring functions, and therefore every practical implementation of the top-k/w system automatically supports the behavior of a Boolean system without any further modifications. This allows us to have Boolean and top-k/w subscriptions simultaneously active in a same top-k/w system, which is very important for the future acceptance of the top-k/w matching model.

**Theorem 2.1.** *The Boolean system is a special case of the top-k/w system where for every* $s \in \mathbf{S}$ *the parameter* $w_s \to 0$ *and its scoring function* $u_s$ *is defined as follows*

$$u_s(p) \stackrel{def}{=} \begin{cases} bs(u_s) & \textit{iff } m_s(p) = \top \textit{ or} \\ ws(u_s) & \textit{otherwise,} \end{cases} \tag{2.78}$$

*where* $m_s$ *is the matching function of* $s$ *in the Boolean system.*

*Proof.* To prove this theorem we have to show that such a top-k/w system would deliver the same set of publications to each subscriber as the corresponding Boolean system. From expressions (2.62) and (2.63) we see that every publication which matches a subscription in the Boolean system will also match the corresponding subscription in the top-k/w system, because there are no publications with better scores at any position within the subscription window. Therefore, the matching process in the top-k/w system is time-independent as in the corresponding Boolean system. In the Boolean system, see expression (2.40) and (2.45), every publication which matches an active subscription must be delivered to the subscriber in time $\leq \delta_p$ after its publishing. Analogously, in the top-k/w system such delivery is also required because every such publication is at the moment of its publishing active, within the subscription window, and not yet delivered according to expressions (2.67) and (2.75). As the safety properties of both systems forbid delivery of other publications, we have proved the theorem. □

## 2.5 Related Work

In this section we survey related work in the area of publish/subscribe system specification and real-time system specification.

### 2.5.1 Formal Specification of Boolean Publish/Subscribe Systems

Mühl, Fiege and Gärtner [89, 144] introduced a formal specification of publish/subscribe systems using Linear-time Temporal Logic (LTL) [136]. Later, Tanner and Mühl [148] presented a strictly propositional[3] formal specification of publish/subscribe systems extended with message completeness guarantees. Finally, Mühl, Fiege and Pietzuch [147] updated the formal specification from [89, 144].

Podnar [167] followed [60] in defining and describing publish/subscribe system properties, but her research was mainly focused on enabling the mobility of clients in publish/subscribe systems. Datta, Gradinariu, Raynal and Simon [72] informally stated liveness and safety for static subscriptions, and with an assumption of global time in the system, which we also follow in this thesis. Courtenage [67] used $\lambda$-calculus for describing event types in publish/subscribe systems, but did not address its correct behavior.

Baldoni, Tucci, Piergiovanni, Virgillito and others [21, 18] presented a formal framework of a distributed computation based on a publish/subscribe system with two types of delays in the system, namely the subscription/unsubscription delay and the diffusion delay. They also formally specified legality, validity and liveness properties. In addition, they mentioned three classes of notification persistence: 0-persistence, $\triangle$-persistence and $\infty$-persistence. The Boolean system has 0-persistence since publications expire soon after being published, whereas the top-k/w system has $\triangle$-persistence, where $\triangle$ is dynamic because a matching publication can be delivered at any point in time while within a subscription window.

Our definition of the Boolean system is the most similar to the definition of the publish/subscribe system without advertisements and self-stabilization from [144], i.e. to the simple event system from [89, 147]. This group of authors defined safety as "what should happen", while we defined the property based on "what should not happen". If we rewrite expressions (2.43), (2.41) and (2.42) following their approach, we will get the following expression:

$$\begin{aligned}\forall y \in \boldsymbol{C}, \forall p \in \boldsymbol{P} : \Box\{&\text{notify}(y,p) \Rightarrow \\ &[\widehat{\Box}\neg\text{notify}(y,p)] \wedge [\exists x \in \boldsymbol{C} : \widehat{\blacklozenge}_{\leq \delta_p}\text{publish}(x,p)] \wedge \\ &[\exists s \in S_y^A : (\widehat{\blacklozenge}_{\leq \delta_s} \textit{is-subscribed}(y,s)) \wedge (p \prec s)]\}. \end{aligned} \tag{2.79}$$

The bounded operators in the upper expression are due to the delay in processing of events publish and unsubscribe, see Example 2.1. Additionally, in our specification of the Boolean system we use logical quantifiers and MITL with both past and future operators, whereas they have used partial quantification and LTL with point-based semantics and variables of the system instead of past operators. By ignoring

[3] In other words, they do not use universal and existential quantification.

these delays and again following their approach, safety (2.79) easily turns into the following expression:

$$\begin{aligned}&\Box\{\text{notify}(y,p) \Rightarrow \\ &\quad [\widehat{\Box}\neg\text{notify}(y,p)] \wedge [\exists x \in \boldsymbol{C} : p \in P_x^P] \wedge [\exists s \in S_y^A : p \prec s]\}.\end{aligned} \tag{2.80}$$

Safety (2.80) states that a publication should never be delivered to a subscribed client more than once, that a delivered publication must have been published by a client in the past, and that a publication should only be delivered to a client if it matches one of the client's active subscriptions. If we ignore processing delays, safety (2.80) is analogous to the safety in our specification (i.e. expressions (2.43), (2.41) and (2.42)), but while the former is almost identical as its counterpart in [144, 89, 147], the latter is closer to the actual definition of the safety property since it defines "what should not happen" [121, 9]. In addition, our liveness property, which is defined in expression (2.40), is drastically different than their definitions from [144, 89] and from [147], respectively:

$$\begin{aligned}&\Box\{\text{subscribe}(y,s) \Rightarrow \\ &\quad [\widehat{\Diamond}\Box(\text{publish}(x,p) \wedge p \prec s \Rightarrow \Diamond\text{notify}(y,p)) \vee \big(\widehat{\Diamond}\text{unsubscribe}(y,s)\big)]\} \text{ and}\end{aligned} \tag{2.81}$$

$$\begin{aligned}&\Box\{\Box s \in S_y^A \Rightarrow \\ &\quad [\widehat{\Diamond}\Box(\text{publish}(x,p) \wedge p \prec s \Rightarrow \Diamond\text{notify}(y,p))]\}.\end{aligned} \tag{2.82}$$

Liveness (2.81) states that if a client subscribes to a subscription, then either there exists a future point in time after which every publishing of a matching publication will lead to its delivery to the client, or there exists a future time when the client unsubscribes from the subscription. Similarly, liveness (2.82) states that if a client activates a subscription and does not unsubscribe from it in future, then there exist a future point in time after which every published matching publication will be delivered to the client. These definitions of the liveness property do not include the case when a client activates a subscription and receives matching publications until it cancels this subscription, which is a very frequent case in practice. According to these definitions, if a client will unsubscribe in the future, the system does not have to deliver any publication to it in a period of time which precedes the unsubscribe event. In addition, the authors mention finite processing delays of subscriptions and publications in a publish/subscribe system, but the maximal durations of these delays are not restricted as in our specification.

### 2.5.2 Formal Specification of Real-Time Systems

In the last twenty years, formal description and analysis of real-time systems has become an increasingly important research topic. This has resulted, among other things, in the development of several temporal logics that are able to express quantitative requirements, which are called real-time logics [15, 16]. There are two main groups of real time logics: branching time (e.g. [11]) and linear time logics (e.g. [200, 118, 16, 13, 177, 43], where the latter are a subset of the former. Linear time logics are based on LTL [136]. Among them, MTL is one of the most popular [90, 157, 156, 169, 78].

It is well known that over the interval-based semantics, the satisfiability and model checking problems

for MTL are undecidable, i.e. they have undecidable satisfiability and model-checking problems [108]. As a consequence, various restrictions on MTL exist (e.g. [174, 110, 199, 13]). There are many results in literature regarding decidability (e.g. [173, 15, 13, 109, 156, 157, 158]) and expressiveness of these restrictions (e.g. [173, 109]). In the last fifteen years, it is believed that some very small fragments of MTL are undecidable. Alur, Feder and Henzinger [13] showed that MITL is decidable, and Alur and Henziger[14] also showed decidability for MITL with past operators. Ouaknine and Worrell recently showed that, on the contrary, some substantial and important fragments of MTL are decidable [156, 157, 158]. D'Souza and Prabhakar showed that MTL with interval-based semantics is strictly more expressive than MTL with point-based semantics, for both future [169] and past [78] temporal operators.

The formal definitions of safety and liveness properties are given in [9]. In the same paper the authors have proven the *decomposition theorem*, i.e. that every property can be written as the intersection of safety and liveness. Liveness properties (2.40) and (2.67) are defined as response properties [108], while safety properties (2.43), (2.41), (2.42), (2.70), (2.68) and (2.69) are defined as invariant properties [190]. A survey of safety and liveness can be found in [116].

# CHAPTER 3

## Efficient Top-k/w Processing over Data Streams

The terminology we use in this chapter is generally accepted in the stream processing community and is different from the one which is usually used by the publish/subscribe community. Since this chapter is related to data stream processing systems we follow their terminology. In other chapters we follow the terminology of the publish/subscribe community. In Table 3.1 we see the comparison of these two terminologies. Besides these differences in terminology, there are also some conceptual differences between stream processing applications and publish/subscribe systems as the number of clients in the former is relatively smaller and the queries are much more complex than those of the latter [206].

## 3.1 Introduction

Data stream processing has become an integral part of many applications in domains such as wireless sensor networks, stock trading, and network monitoring that require data processing in real-time, and cope with high rates of data object publication while queries are mostly static and continuous. Applications such as the Sensor Web [17], or RSS filters and aggregators [180] are typically distributed, and span over numerous data sources in wide area networks. As such data sources typically generate data at high rates, it is imperative to process incoming data streams close to their sources to avoid network overload, and filter out superfluous data in order to deliver information of interest in real-time and prevent information overload by end-users.

A data stream is a continuous and possibly infinite sequence of data objects that arrive in an arbitrary order into a processing system to be processed in real-time [98]. In such a setting, a data object from an input data stream is matched against a set of continuous queries upon its arrival into a processing system, and is subsequently inserted into the output stream associated with a continuous query if the object matches the query. In this chapter, we study a particular type of continuous queries which monitor top-k data objects over sliding windows (*top-k/w queries*). Sliding windows are used to restrict the temporal

Table 3.1: Differences in the terminologies used by the stream processing and publish/subscribe communities.

| Stream Processing Community Terminology | Publish/Subscribe Community Terminology |
|---|---|
| data object | publication |
| continuous query | subscription |
| data stream | incoming publications |
| query updates | subscription activations and cancellations |
| query result stream | delivered publications |

scope of query processing in the absence of explicit deletions of data objects [143], and are commonly defined as either the number of most recent data stream objects (count-based windows) or time intervals (time-based windows). The parameter $k$ defines the number of matching data objects restricting it to top-k objects within a sliding window according to an arbitrary scoring function. By adjusting the parameter $k$ to sliding window size, the processor controls the rate of data insertion into the output stream, and can thus prevent data overload.

Our goal is to design a data stream processing system which produces resulting data streams associated with multiple top-k/w queries in an efficient manner, i.e. with a minimal memory footprint and processing time. In the sliding window model, if a data object is not a top-k object within a window at the time of its arrival in the system, it can become one later on when better-ranked data objects are dropped from the window while, at the same time, only objects with lower ranks are arriving. Therefore the processing engine has to store in memory all data objects within the window that have the potential to become top-k objects in the future. A straightforward approach would maintain all sliding window objects in memory, however, since we are targeting environments that cannot store all data objects within the window in memory, there is a need to reduce the set of maintained top-k data candidates, especially in scenarios when $k$ is much smaller than the window size. Moreover, the size of the top-k candidate set directly influences processing performance.

In this chapter, we define a generic top-k/w data stream processing model that is independent of both data representation and scoring function for evaluating the relevance of data objects to a continuous query. The model serves as the basis for a number of algorithms for top-k/w processing over certain streams that define, analyze, and evaluate experimentally. The algorithms apply different approaches for the maintenance of top-k candidate objects in memory. Two algorithms are deterministic: The Strict Candidate Pruning Algorithm (SA) continuously maintains a minimal set of top-k objects in a k-skyband and is based on the algorithm originally proposed in [32], while the Relaxed Candidate Pruning Algorithm (RA) periodically prunes non-candidate objects from a k-skyband to offer improved processing performance at the expense of increased memory consumption. RA is a novel deterministic algorithm and represents our original contribution in this chapter. Furthermore, we propose a probabilistic algorithm which uses a probabilistic criterion to decide upon the arrival of a new data stream object whether it has the potential to become a top-k object in future, and generates approximate results to top-k/w queries. The Probabilistic Algorithm (PA) is an improved version of our original algorithm initially presented

in [172], and can only be used for processing random-order data streams [51]. Our experimental results show that PA significantly outperforms both SA and RA in terms of memory consumption and processing performance, while the observed PA error rate is very low and significantly smaller than the theoretical upper bound. We conclude that it is highly beneficial to employ the light-weight and efficient probabilistic solution in resource-constrained environments with random-order data streams when an application can tolerate low error rates.

Additionally, we use the idea of employing a buffer of recent stream objects [32] to further improve SA and RA implementations by delaying the insertion of recent objects into the k-skyband. Buffer implementation uses the principles introduced in PA and performs the function of a probabilistic filter (PF). When such a buffer is combined with either SA or RA, the resulting algorithms are also deterministic. As our experimental study shows, the novel deterministic algorithms SAPF and RAPF benefit from the probabilistic criterion and offer significantly improved performance compared to both SA and RA, and can be recommended for applications requiring exact processing results.

Although data stream processing has been a particularly active research area in the last years [52, 97, 4, 91], solutions for efficient processing of top-k/w queries are still lacking. The existing papers [117, 141, 71, 143, 32] do not offer a generalized model for top-k/w processing, but rather focus on special top-k problems (e.g. k-NN) and specific data representations. The algorithm presented in [32] is most relevant to our work and serves as the basis for our implementation of SA. The algorithms presented in [141, 143] are defined for top-k/w queries using aggregation scoring functions and store in memory all data objects within the window, which is inefficient and unfeasible in the case of data sources with high streaming rates. The approximate algorithm presented in [117] is most relevant to our PA, however it is also tailored to k-NN queries. Thus this chapter represents an extensive report which evaluates and compares deterministic and probabilistic algorithms for top-k/w data stream processing.

The chapter is organized as follows. Section 3.2 presents a formal top-k/w data stream processing model and gives a formal explanation of different issues affecting the practical implementation of the processing model. In Section 3.3 we present two deterministic algorithms SA and RA, and introduce PA in Section 3.4. The extended versions of SA and RA are reported in Section 3.5, while a more general situation with multiple queries is discussed in Section 3.6. Section 3.7 analyzes space and time complexity of the proposed algorithms. In Section 3.8 we provide an extensive experimental evaluation of all algorithms using synthetic and real data sets. A survey of related work in the field of top-k/w queries on data streams is given in Section 3.9 and Section 3.10 concludes this chapter.

## 3.2 Top-k/w Data Stream Processing Model

Figure 3.1 depicts the architecture of our data stream processor which is in accordance with the generic architecture proposed in [98]. As we can see in the figure, a single append-only data stream represents input to the data stream processor, while each incoming data object has object-specific expiry. This implies that data objects cannot change after entering the system, while the time of object expiry is independent from its time of arrival. For example, we are considering preprocessed sensor readings, either from a sin-

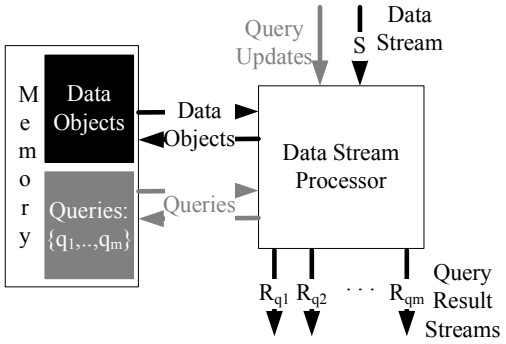



Figure 3.1: Data stream processor architecture.

gle sensor or aggregated readings from wireless sensor networks, or RSS feeds as valid input data streams for our stream processor. The processor accepts top-k/w queries that may be activated and canceled at will. We assume queries reference future data objects, i.e. objects entering the system after query activation, and cannot reference past objects that have still not expired at the time of query activation. The processor memory stores both active continuous queries and a set of data objects from the input stream needed for top-k/w processing. Notice that each incoming data object is seen only when entering the system unless it is explicitly stored in memory. As we assume memory size is relatively small compared to the size of data within the stream window, only a restrictive subset of data objects can be stored in memory. Furthermore, since the size of such a data set directly influences processor performance, in this section we formally define the set of objects that are necessary for continuous top-k/w processing.

In this section we first present a data stream processing model for continuous top-k/w processing, and introduce the terminology and notation used throughout the chapter. Without loss of generality, the model is built assuming count-based sliding windows associated with continuous queries and may be extended in a straightforward manner to support time-based sliding windows. Second, we provide a formal problem definition which identifies data stream objects maintained in memory which are necessary for continuous top-k/w processing. Third, we briefly discuss the implications of the presented model on processor performance in comparison to related work.

### 3.2.1 Model Definition

**Definition 3.1** (Data Stream Object)**.** Let $t_o^A$ and $t_o^E$ be two points in time such that $t_o^A < t_o^E$, and let $c_o$ be the object content. We define a triple $o = \{c_o, t_o^A, t_o^E\}$ as a data stream object.

A data stream object is represented by its content, e.g. it may be a point from a multidimensional attribute space or an unstructured textual document. The object has an associated *time of appearance* $t_o^A$ denoting a point in time when object $o$ enters the system, and *time of expiry* $t_o^E$, which is object-specific and unrelated to $t_o^A$. We assume the object content is certain, while its content and time of expiry do not change after entering the processor.

**Definition 3.2** (Data Stream)**.** A data stream $S$ is an infinite and ordered set of data stream objects $S = \{o_1, \ldots, o_i, \ldots\}$ ordered by their time of appearance such that $\forall(o_i, o_j \in S) : (i < j) \Leftrightarrow (t_{o_i}^A < t_{o_j}^A)$,

where $i,j \in \mathbb{N}$ are *sequence numbers* of $o_i$ and $o_j$, respectively.

In other words, data stream objects are unique in their time of appearance implying that a single object may appear at a point in time. Therefore, all objects from a data stream can be ordered by their time of appearance.

Each data object that has entered the system is either active or inactive at a point in time $t$. Thus we define a set of active data objects at $t$ which contains all objects that have appeared before $t$, and have still not expired at $t$.

**Definition 3.3** (Active Data Stream Objects)**.** Let $t$ be a point in time, and let $S$ be an incoming data stream. We define the set of active data stream objects $A(t) \subseteq S$ at a point in time $t$ as

$$A(t) \stackrel{\text{def}}{=} \{o : (o \in S) \wedge (t_o^A < t \leq t_o^E)\}. \tag{3.1}$$

Apart from being ordered by their time of appearance, data stream objects may be evaluated based on their content. Thus we define a scoring function for calculating object-specific scores.

**Definition 3.4** (Scoring Function)**.** Let $S$ be an incoming data stream. We define the scoring function $u : S \mapsto \mathbb{R}$ as any function which assigns a score $\forall o \in S$.

Scoring functions are application specific and their explicit definition depends completely on the application scenario in which the processor is used. In a typical application scenario, the scoring function will depend exclusively on an object's content, i.e. $u(o) = f(c_o)$.

Object ranking is vital for the implementation of top-k query processing and we use object scores to rank objects primarily based on their content. Some scoring functions assign ranks to object scores in descending order such that objects with higher scores are ranked higher than objects with lower scores, while other types of scoring functions assign ranks to object scores in ascending order. To enable object ordering by either decreasing or increasing scores, we define a Boolean function for comparing object scores. As data objects are static and cannot change over time, their scores also do not change over time, however, their ranks may change as new objects appear in the system.

**Definition 3.5** (Score Comparator)**.** Let $u$ be a scoring function, let $S$ be an incoming data stream, and let $o, o' \in S$. We define the score comparator $\rhd$: $S \times S \mapsto \{\top, \bot\}$ of $u$ as the following Boolean function

$$o \rhd o' \stackrel{\text{def}}{=} \begin{cases} \top & \text{iff } u \text{ assigns a higher rank to } o \text{ than to } o', \\ \bot & \text{iff } u \text{ assigns a lower rank to } o \text{ than to } o'. \end{cases} \tag{3.2}$$

The score comparator compares two data objects by their scores such that the result is $\top$ when a higher rank is assigned to the first argument ($o$) than to the second argument ($o'$), or $\bot$ otherwise. For example, $o_1 \rhd o_2 \rhd \ldots o_k = \top$ denotes a ranked list of $k$ data stream objects. Obviously, $o$ and $o'$ are non-commutative for $\rhd$, and we can define an inverse score comparator $\lhd$: $S \times S \mapsto \{\top, \bot\}$ as follows: $o \lhd o' \stackrel{\text{def}}{=} \neg(o \rhd o')$.

We assume unique scores can be computed for all data objects in $S$ such that all objects may be ranked uniquely. If this is not the case, the scoring function can be modified to include time of object appearance which is unique for each object, i.e. $u(o) = f(c_o, t_o^A)$. For example, if the same score is assigned to two different data stream objects, we can modify the scoring function to give preference and a higher rank to more recent objects.

**Definition 3.6** (Higher Ranked Objects)**.** Let $u$ and $\rhd$ be a scoring function and an associated score comparator, let $S$ be an incoming data stream, let $O \subseteq S$ be a set of data stream objects, and let $o \in S$ be an incoming data object. We define a set of objects $H_{\rhd}(o, O) \subseteq O$ composed of data objects from $O$ that, according to $u$ and $\rhd$, have higher ranks (i.e. better scores) than $o$. More formally,

$$H_{\rhd}(o, O) \stackrel{\text{def}}{=} \{o' : (o' \in O) \wedge (o' \rhd o)\}. \tag{3.3}$$

We denote the set $H_{\rhd}(o, O)$ by $O \rhd o$, and read it as "objects from $O$ with higher rank than $o$".

Hereafter we identify characteristics of top-k/w queries supported by our data stream model. Each query is defined as a scoring function associated with a score comparator and additional two parameters, the size of the sliding window $n$, and parameter $k$—the number of top-k objects within the window. Furthermore, as queries are continuous, they have a predefined time of activation and cancellation.

**Definition 3.7** (Top-k/w Query)**.** Let $u_q$ and $\overset{q}{\rhd}$ be a scoring function and associated score comparator. Let $k_q, n_q \in \mathbb{N}$ and let $t_q^A$ and $t_q^C$ be two points in time such that $t_q^A < t_q^C$. We define a sextuple $q = \{u_q, \overset{q}{\rhd}, k_q, n_q, t_q^A, t_q^C\}$ as a continuous top-k query over a count-based sliding window (*top-k/w query*).

The scoring function $u_q$ and associated score comparator $\overset{q}{\rhd}$ are defined per query in our model, but in an actual application scenario, they may also be defined uniquely for all queries. However, for the purpose of generality, we assume that scoring functions and score comparators are defined per query. Parameter $n_q$ is the size of count-based query window, and $k_q$ is the number of top objects within the window that have to be inserted in the query result stream. A point in time $t_q^A$ is the *time of query activation*, while $t_q^C$ is the *time of query cancellation*.

We are particularly interested in distance, aggregation, and relevance functions as scoring functions commonly used in application domains that we envision for our processor. Distance and aggregation functions are typically used in applications processing structured data, while relevance functions are frequently used by search engines processing unstructured textual data. An example aggregation function is a weighted-sum function which assigns scores to data objects represented as points in an attribute space according to the sum of their weighted attribute scores. An example distance function is the Euclidean distance between points representing data objects in an attribute space. Finally, an example relevance function is the cosine of the angle between two vectors representing unstructured textual documents in a vector-space.

**Definition 3.8** (Data Objects within a Query Window)**.** Let $q$ be a query, let $t$ be a point in time when $q$ is active, let $S$ be an incoming data stream, and let $o_z$ be the last object that has appeared in the system at

$t$ such that $\nexists o \in S : (t > t_o^A > t_{o_z}^A)$. We define the set of data objects within the query window $W_q(t) \subseteq S$ of $q$ at $t$ as follows:

$$W_q(t) \stackrel{\text{def}}{=} \{o_i : (o_i \in S) \wedge (t_q^C \geq t > t_{o_i}^A > t_q^A) \wedge (i > z - n_q)\}, \tag{3.4}$$

where $i$ and $z$ are sequence numbers of $o_i$ and $o_z$, respectively.

In other words, a data object is within the query window at a point in time $t$, if the object enters the processor after query activation, and is among the $n_q$ most recent data objects. We are therefore supporting *ad-hoc queries referencing future objects*. Note that, as new objects appear in $S$, old data objects are dropped from the query window. We denote by $t_o^D$ the point in time when $o$ is dropped from the query window because a new object $o_a$ has just appeared and expelled $o$ from the window, i.e. $t_a^A = t_o^D$. Contrary to time-based windows, the exact moment when $o$ is dropped from the window cannot be known in advance for count-based windows, because it is the consequence of $o_a$'s appearance.

The following definition specifies the set of *valid data objects* that have the potential to become top-k/w objects.

**Definition 3.9** (Valid Data Objects for a Query)**.** Let $q$ be a top-k/w query, and let $t$ be a point in time when $q$ is active. We define the set of valid data objects $V_q(t) \subseteq W_q(t)$ for $q$ at $t$ as

$$V_q(t) \stackrel{\text{def}}{=} \{o : (o \in W_q(t)) \wedge (o \in A(t))\}. \tag{3.5}$$

In other words, valid data objects at $t$ are all objects that are both active and within the query window at $t$. Let us now discuss three possible scenarios that lead to object invalidation: (1) an active object is dropped from the query window, (2) an object becomes inactive while it is within the query window, and (3) an object becomes inactive and is dropped from the window simultaneously. An active object $o$ is dropped from the window if $t_o^A < t_o^D < t_o^E$, and becomes invalid at $t_o^D$ since it is no longer of interest to $q$. An inactive object remains within the query window when $t_o^A < t_o^E < t_o^D$, and becomes invalid at $t_o^E$ even if it is still within the window. An object simultaneously becomes inactive and is dropped from the window if its active period equals the window size (i.e. $t_o^A < t_o^E = t_o^D$), and is obviously considered valid until $t_o^E = t_o^D$.

Next, we define top-k objects with the query window at $t$.

**Definition 3.10** (Top-k Objects Within a Query Window)**.** Let $q$ be a top-k/w query, and let $t$ be a point in time when $q$ is active. We define the set of top-k data objects within the query window $T_q(t) \subseteq V_q(t)$ for $q$ at $t$ as follows:

$$T_q(t) \stackrel{\text{def}}{=} \{o : (o \in V_q(t)) \wedge (|W_q(t) \stackrel{q}{\rhd} o| < k_q)\}. \tag{3.6}$$

In other words, a valid data object $o$ is a top-k object for a top-k/w query at $t$ if at $t$ there are less than $k_q$ objects with higher rank than $o$ among all objects within the current query window. Note that the set of top-k objects is continuously being updated and changes over time. Next, we define under which conditions a data object must be reported as a query result and define a set that forms the correct answer to a top-k/w query.

**Definition 3.11** (Query Result Set)**.** Let $q$ be a top-k/w query, and let $S$ be an incoming data stream. We define the query result set $R_q \subseteq S$ for $q$ as follows:

$$R_q \stackrel{\text{def}}{=} \{o : (o \in S) \wedge \exists t(o \in T_q(t))\}. \tag{3.7}$$

A result set associated with top-k/w query $q$ is deterministically correct iff it contains all data objects from $S$ that are top-k objects within the query window at any $t$, such that $t_q^A < t \leq t_q^C$. Finally, we define the correct result stream associated with a top-k/w query.

**Definition 3.12** (Query Result Stream)**.** Let $q$ be a top-k/w query. We define a result stream $S_q$ for $q$ as an ordered set of data objects $S_q = \{o_1, o_2, \dots\}$ for which the following holds: $\forall(o_i, o_j \in R_q) : (i < j) \Leftrightarrow (t_{o_i}^R < t_{o_j}^R)$, where $i,j \in \mathbb{N}$ are sequence numbers of $o_i$ and $o_j$ in $S_q$, while $t_{o_i}^R$ and $t_{o_j}^R$ are points in time when $o_i$ and $o_j$ are inserted into $q$'s result stream, i.e. when they become elements of $R_q$.

According to the previous definition, the processor produces an associated stream of objects for each top-k/w query in the system, i.e. queries $q_1, \dots, q_m$ in Figure 3.1.

### 3.2.2 Problem Statement

Let us now discuss when an object $o$ is inserted into a query result set, i.e. $t_o^R$. According to Definition 3.10, a valid object becomes a top-k query object when there are less than $k_q$ higher ranked objects within a window. This can happen under the following circumstances:

- at $t_o^A$ when $o$ enters the processor iff there are less than $k_q$ higher ranked objects within the query window, and
- at a later point in time $t_o^A < t \leq t_o^E$ when one of top-k objects is dropped from the query window and $o$ has the highest rank among all other objects within the window that are not top-k objects.

The implementation of the first scenario is quite straightforward: Each incoming data object is compared against the list of current top-k query objects, and, in case its rank is higher than the rank of the last top-k object, it is added into the query result set. This requires continuous maintenance of top-k objects in memory, because, although these objects have already been inserted into the query result stream, they are continuously compared to arriving objects. Additionally, we need to detect empty slots in the top-k list to implement the second scenario.

The second scenario requires a more elaborate solution as it has already been recognized in [143, 32, 114]: The processor needs to instantly fill the slot of an invalidated top-k object with an object that currently has the best rank among non top-k objects within the query window. Therefore, the processor needs to store additional *candidate data objects* in memory that have the potential to become top-k objects in the future. The most trivial solution would be to maintain all data objects within the query window in memory, and discard them later on when they are dropped out of the query window, as it has been suggested in [141, 143]. However, as query window size can in general be much larger than $k$, and as we

assume that memory is limited, our goal is to minimize the number of candidate objects maintained in memory.

Hereafter we define a property that can be used to detect that a data object is not a candidate query object.

**Definition 3.13** (Domination)**.** Let $q$ be a continuous top-k/w query, let $S$ be a data stream in the system, and let $o, o' \in S$. We define *domination* $\overset{q}{\blacktriangleright}$: $S \times S \mapsto \{\top, \bot\}$ as the following Boolean function:

$$o \overset{q}{\blacktriangleright} o' \overset{\text{def}}{=} (o \overset{q}{\triangleright} o') \wedge (t_o^A > t_{o'}^A). \tag{3.8}$$

In other words, a data object *dominates* another object for a given query iff the former object is more recent, i.e. younger, and has a higher rank for the query than the latter object. We can also write $o \overset{q}{\blacktriangleleft} o' \overset{\text{def}}{=} (o' \overset{q}{\blacktriangleright} o)$ which states that $o$ is *dominated by* $o'$.

**Definition 3.14** (Data Object Dominators for a Query)**.** Let $q$ be a continuous top-k/w query, let $t$ be a point in time when $q$ is active, let $S$ be a data stream in the system, and let $o \in W_q(t)$. We define the set of dominators of $o$ for $q$ at $t$ as follows

$$D_q(t, o) \overset{\text{def}}{=} \{o' : (o' \in W_q(t)) \wedge (o' \overset{q}{\blacktriangleright} o)\}. \tag{3.9}$$

The set $D_q(t, o)$ contains all objects that dominate $o$ for $q$ at $t$, and in the rest of the chapter, we denote this set as $W_q(t) \overset{q}{\blacktriangleright} o$ and read it as "objects within the window of $q$ that dominate $o$ at $t$".

The following two theorems define when a data object can no longer affect the query result stream and is therefore not a candidate query object.

**Theorem 3.1.** *Let $q$ be a query in the system, and let $t$ be a point in time when $q$ is active. A data object $o$ that is valid for $q$ at $t$ and cannot become its top-k object at any future point in time $t' > t$ if at $t$ it has $k_q$ or more dominators. More formally,*

$$\forall o \in V_q(t) : (|W_q(t) \overset{q}{\blacktriangleright} o| \geq k_q) \Rightarrow (\nexists t' > t : o \in T_q(t')). \tag{3.10}$$

*Proof.* Assume to the contrary that $\exists t' > t : o \in T_q(t')$ and that $|W_q(t) \overset{q}{\blacktriangleright} o| \geq k_q$. Using expressions (3.8) and (3.9) we have $|W_q(t) \overset{q}{\blacktriangleright} o| \geq k_q \Rightarrow (|W_q(t) \overset{q}{\triangleright} o| \geq k_q)$ which is valid $\forall t' : (t_o^D \geq t' > t > t_o^A)$ and is in contradiction with expression (3.6). □

In other words, a data object $o$ is not a candidate data object for $q$ if, at any point in time when $o$ is valid, it becomes *dominated* by $k_q$ or more than $k_q$ objects within the query window, i.e., there are $k_q$ younger objects with higher rank than $o$. In the rest of the chapter we say that such a data object is *dominated for query $q$*, and in case $o$ is dominated by less than $k_q$ objects, we say the object is *non-dominated for query $q$*.

**Theorem 3.2.** *Let $q$ be a query in the system, and let $t$ be a point in time when $q$ is active. A data object $o$ that is valid for $q$ at $t$ cannot become its top-k object at any future point in time $t' > t$, if at $t$ there are*

*$k_q$ or more data objects with higher rank than o within the window of q that will stay inside the window until q is canceled. More formally,*

$$\forall o \in V_q(t) : (\exists O : (O \subseteq (W_q(t) \overset{q}{\triangleright} o)) \wedge (|O| \geq k_q) \wedge (\forall o' \in O : (t^D_{o'} > t^C_q))) \Rightarrow \\ (\nexists t' > t : o \in T_q(t')). \quad (3.11)$$

*Proof.* Assume to the contrary that $\exists O : (O \subseteq (W_q(t) \overset{q}{\triangleright} o) \wedge (|O| \geq k_q)$ and that $\exists t' > t : o \in T_q(t')$. As $t^D_{o'} > t^C_q$, $o'$ will stay within the query window up to the point in time $t^C_q$, $\forall o' \in O$, and thus we can write $\forall t' : (t^C_q \geq t' > t) \Rightarrow (|W_q(t) \overset{q}{\triangleright} o| \geq |O| \geq k_q)$, which is in contradiction with expression (3.6). □

In other words, a data object $o$ is not a candidate data object for $q$ if, at any point in time when $o$ is valid, there are $k_q$ or more objects with higher rank than $o$ within the query window that will remain within the window until $q$ is canceled. This property is useful for discarding objects when the duration of the query window is comparable to object activity period.

Additionally, we show in the following lemma that a dominated object can dominate only already dominated objects, and therefore such objects are truly irrelevant for a query and may safely be *dereferenced*. A data object which is dereferenced by all queries in the system may safely be discarded from processor memory.

**Lemma 3.3.** *Let q be a query in the system, and let t be a point in time when q is active. A data object o that is dominated for q at t can dominate only objects which are already dominated at t. More formally,*

$$\forall o, o' \in W_q(t) : (o \overset{q}{\blacktriangleright} o') \wedge (|W_q(t) \overset{q}{\blacktriangleright} o| \geq k_q) \Rightarrow |W_q(t) \overset{q}{\blacktriangleright} o'| \geq k_q. \quad (3.12)$$

*Proof.* Assume to the contrary that $\exists o' \in W_q(t)$ such that $o \overset{q}{\blacktriangleright} o'$, $|W_q(t) \overset{q}{\blacktriangleright} o| \geq k_q$ and $|W_q(t) \overset{q}{\blacktriangleright} o'| < k_q$. From expression (3.9) we have $\forall o'' \in (W_q(t) \overset{q}{\blacktriangleright} o) : (o'' \overset{q}{\triangleright} o \overset{q}{\triangleright} o') \wedge (t^A_{o''} > t^A_o > t^A_{o'})$ and then $|W_q(t) \overset{q}{\blacktriangleright} o'| \geq k_q$. □

Both Theorem 3.1 and Theorem 3.2 define the properties of objects that are irrelevant for a query and can be used to discard dominated objects as unsuitable candidate objects. However, Theorem 3.1 is more usable in practice because it can be applied to discard dominated data objects at any point in time, while Theorem 3.2 is useful shortly before query cancellation or when the duration of the query window is comparable to object activity period. Moreover, as it is difficult to predict the time of query cancellation in practice which is required for the implementation of Theorem 3.2, we apply Theorem 3.1 and Lemma 3.3 as the basis for defining two deterministic top-k/w processing algorithms presented in Section 3.3.

### 3.2.3 Discussion

In this subsection we discuss three issues: an alternative to our top-k/w processing model presented in Section 3.2.1, required modifications of our model to support object deletions and time-based windows.

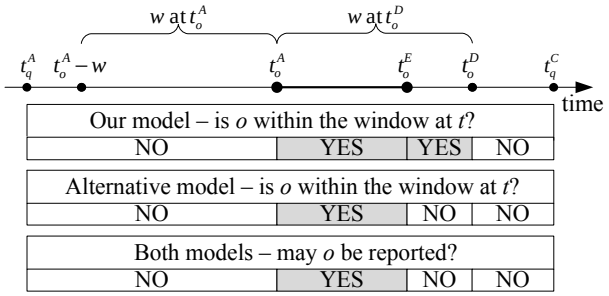



Figure 3.2: Comparison of our and the alternative top-k/w processing model

The alternative model defines top-k objects within the query window in the following way:

$$T'_q(t) \stackrel{\text{def}}{=} \{o : (o \in V_q(t)) \wedge (|V_q(t) \overset{q}{\rhd} o| < k_q)\}. \tag{3.13}$$

This is the alternative to our Definition 3.10 and the corresponding expression (3.6).

Please note that the second part of the previous expression differs for the two models since the set $W_q(t)$ used in our model is replaced with the set $V_q(t)$ in the alternative model. In other words, when determining whether $o$ is a top-k objects, the alternative model compares $o$ only with objects that are *valid* for the query at $t$, while our model compares $o$ with all data objects *within the query window* at $t$. Note that, according to both definitions, a data object has to be valid to become a top-k object, however, an object competes only against valid objects within the window in the alternative model, while in our model it is compared against all objects within the window, even though some of these objects might already be expired. Our reasoning is that each object within the window, although expired, still holds its position within the window, and cannot be discarded as it posses important information about the characteristics of objects within the query window.

**Example 3.1.** The aforementioned slight difference in the definitions can have an enormous effect on system performance when there are many inactive objects within the query window, i.e., when object lifetimes are shorter than the size of a query window. This scenario is shown graphically in Figure 3.2. A data object $o$ enters the query window at $t_o^A$ and expires at $t_o^E$, although its $t_o^D$ follows after $t_o^E$ since the window size is larger than $t_o^E - t_o^A$. For both models, $o$ can be reported as a top-k object only when it is active, i.e. in the period $(t_o^A, t_o^E]$. The difference relates to the question whether $o$ is considered to be within the window or not: The alternative model considers it is within the window only in the period $(t_o^A, t_o^E]$, while our model considers it is within the window during $(t_o^A, t_o^D]$.

Let us now discuss the implications of the alternative definition for top-k/w objects within the query window on processor performance.

**Objects have non-monotonic times of dropping from a query window** since data objects whose lifetime is shorter than the size of query window, i.e. inactive objects within the window, will be dropped from the window at $t_o^E$, while others will be dropped from the window at $t_o^D$.

**It is difficult to support the combination of count-based windows and object-specific expiry** as we cannot know in advance which event precedes another, either object expiry or object dropping from the window due to arrival of a new object. This complicates the implementation of data object discarding because, besides object score, we also have to compare $t_o^A$, $t_o^E$, and $t_o^D$ to determine object dominance.

**The deletion of objects is supported only when all valid objects are stored in memory** since an object that was dominated for a query and thus discarded can later become a top-k object when some of its dominators are deleted.

**An inflation of top-k objects is possible** in an extreme situation when many objects become inactive within the window because $|V_q(t)| \ll |W_q(t)|$ and thus $|V_q(t) \overset{q}{\rhd} o| \ll |W_q(t) \overset{q}{\rhd} o|$. As $|V_q(t) \overset{q}{\rhd} o| < k_q$, this implies that expiring objects, although still within the window, are creating free slots within the top-k list when they expire instead of creating free slots when dropping from the window. This gives the opportunity to newly arriving objects to become top-k objects when entering the processor and thus produces more than $k_q$ objects within the window in the query result stream. In the worst-case scenario when object lifetimes are shorter than object inter-arrival times, all incoming objects would enter the query result stream. Such processor performance is in contradiction with the requirements of top-k/w stream processing.

Hyper-production of top-k objects and lack of deletion are the main drawbacks of the alternative model and the major incentives for redefining the top-k/w data stream processing model. Although the authors in [32, 114] do not provide a fine-grained definition of top-k/w objects within the query window, they mention and discuss some of the previously listed problems, and we can therefore argue their work is based on the alternative model. Let us now discuss modifications of our model to support object deletions.

Following the previous discussion, our model can be extended in a straightforward manner to support data object deletions and thus *dynamic data streams*. Analogous to expired objects within the query window, deleted objects become inactive but remain within the window until they are dropped from the window. Of course, once deleted, the object cannot become a top-k object in future and therefore we may safely delete its content from memory, and store only its identifier associated with object score while it is within the window.

Here we sketch a solution which adds support for object deletions. The model can be extended in the following way: 1) associate a unique identifier *id* with each object, and 2) redefine the content of data streams to carry both insertions of objects $I(o)$ and deletions of previously inserted objects $D(\{id, t_o^E\})$. When a deletion appears at input, the processor needs to update the time of object expiry ($t_o^E$) for a previously defined object with an associated identifier *id*.

Finally, to extend our model with time-based windows we give the analogue of Definition 3.8.

**Definition 3.15** (Data Objects in the Query Window)**.** Let $q$ be a query, and let $t$ be a point in time when $q$ is active. We define the set of data objects within the query window $W_q(t) \subseteq S$ of $q$ at $t$ as follows

$$W_q(t) \overset{\text{def}}{=} \{o : (o \in S) \wedge (t_q^C \geq t_o^A + w_q \geq t > t_o^A > t_q^A\}, \tag{3.14}$$

where $w_q \in \mathbb{R}^+$ is the size of the time-based window of $q$.

## 3.3 Deterministic Algorithms

In this section we present two deterministic algorithms for continuous processing of top-k/w queries on data streams. Both algorithms maintain a k-skyband of data objects for active queries in the system which contains non-dominated objects for each query and enables pruning of dominated data objects. According to Theorem 3.1 and Definition 3.14, when detecting dominated data objects, we compare data objects along two dimensions: the time of object appearance and object score. The comparison is always two-dimensional, and it neither depends on the type of query scoring function, nor dimensionality and representation of the data object content. For this reason, the algorithms we present in this section assume generic top-k/w queries, which are independent of object representation and query definition.

The first deterministic algorithm is the *strict candidate pruning algorithm* which is based on continuous strict maintenance of a k-skyband associated with a query. This algorithm uses dominance counters for k-skyband maintenance as already suggested in [141, 32, 143], and serves as the reference algorithm for our experimental evaluation in Section 3.8. Next, we propose our original deterministic top-k/w processing algorithm entitled the *relaxed candidate pruning algorithm*. It maintains a relaxed k-skyband for each active query and periodically prunes dominated data objects.

Although the presented top-k/w processing model supports multiple queries, to simplify discussion in this section, we present algorithms with respect to a single top-k/w query $q = \{u, \triangleright, k, n, t^A, t^C\}$ if otherwise not explicitly stated. Specifics related to concurrent processing of multiple queries is discussed in Section 3.6. Furthermore, we assume the score comparator $\triangleright$ is defined to assign higher ranks to objects with higher scores.

In this section we first formally define the *k-skyband* and investigate vital properties for the implementation of the two deterministic algorithms. Next, we describe the strict candidate pruning algorithm and conclude this section with the definition of the relaxed candidate pruning algorithm.

### 3.3.1 k-Skyband

We can present data objects from the query window as points in two-dimensional Cartesian space where one axis represents object score and the other time of object appearance. We can arrange these points, and thus also data objects, into skylines and the corresponding skyband. According to [160], a set of points that are dominated by $k-1$ other points is a *k-skyline*, while a *k-skyband* is a set of points that are dominated by less than $k$ other points.

**Example 3.2.** Figure 3.3 depicts a graph with a number of points representing objects from an input data stream in such Cartesian space. It is quite straightforward to detect data objects that dominate a chosen object, or a set of objects dominated by an object using such graphical representation. For example, point $j$ is dominated by points in its upper-right quadrant (points $b$ and $c$), since object $o_j$ is dominated by objects with higher scores that are younger than $o_j$. Analogously, point $j$ dominates all points located in

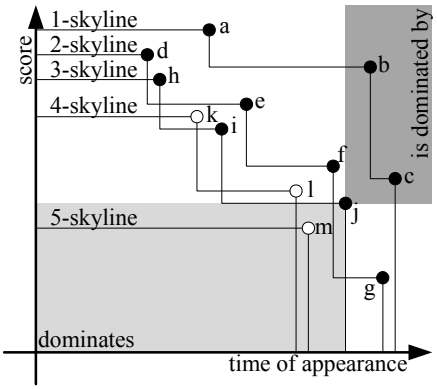



Figure 3.3: An example *k*-skyband.

its lower-left quadrant (point *m*) because an object dominates all objects with lower scores and smaller time of appearance. Additionally, points *a*, *b*, and *c* in Figure 3.3 represent a 1-skyline because they are not dominated by any other points, while point *m* represents a 5-skyline because it is the only point dominated by 4 other points (*b*, *c*, *f*, and *j*). All black points in Figure 3.3 form a 3-skyband, which is composed of the 1-skyline, 2-skyline and 3-skyline.

Hereafter we define the query k-skyband that contains all objects within the query window that are dominated by less than *k* objects (we say they are non-dominated for a query), since, according to Theorem 3.1, candidate objects cannot be dominated by *k* or more than *k* objects.

**Definition 3.16** (Query k-Skyband)**.** Let $t$ be a point in time when a top-k/w query $q$ is active. Then, we define k-skyband $S_q(t) \subseteq W_q(t)$ associated with $q$ at $t$ as the set of data objects from the query window at $t$ which are dominated by less than $k_q$ data objects at $t$. More formally,

$$S_q(t) \stackrel{\text{def}}{=} \{o : (o \in W_q(t)) \wedge (|W_q(t) \overset{q}{\blacktriangleright} o| < k_q)\}. \tag{3.15}$$

The set of objects within the k-skyband contains all top-k objects and objects that have the potential to become top-k in future at a point in time $t$. Therefore, if a query k-skyband is maintained in memory over time, it can be used to deterministically answer the associated top-k/w query. For example, the 3-skyline in Figure 3.3 can be used to answer a top-3 query at a point in time $t$ assuming all presented points are within the query window at $t$. Note the query k-skyband is constantly evolving over time as new objects arrive and existing become dominated by $k$ or more than $k$ objects.

The query k-skyband surely contains the minimal set of candidate data objects. However, it also includes some additional objects that, according to Theorem 3.2, cannot become top-k in future. This is stated formally in the following theorem.

**Theorem 3.4.** *Let q be a query in the system, and let t be a point in time when q is active. The minimal set of objects that can become top-k objects of q at any future point in time $t' > t$ is a subset of the k-skyband*

*associated with q at t. More formally,*

$$\forall(o \in W_q(t)) : (o \notin S_q(t)) \Rightarrow (\nexists t' > t : o \in T_q(t')). \tag{3.16}$$

*Proof.* For a data object $o \in W_q(t)$ that is not a $k$-skyband object of $q$ at $t$, Definition 3.16 gives $|W_q(t) \overset{q}{\blacktriangleright} o| \geq k_q$. From Theorem 3.1 we can write $\nexists t' > t : o \in T_q(t')$. Additionally, for some objects from $S_q(t)$ the condition from Theorem 3.2 can hold, and thus the $k$-skyband is a superset of the the minimal set of objects that can become $q$'s top-k objects. □

Therefore, to produce deterministic answers to a top-k/w query it is sufficient to maintain the query k-skyband over time if we accept slight overhead in memory consumption due to the maintenance of objects that according to Theorem 3.2 cannot become top-k in future. Please note that maintaining such a two-dimensional k-skyband is a much simpler problem than the maintaining of three or more dimensional k-skybands [170].

Let us now discuss a property formally defined in the following lemma that is important for the deterministic algorithms defined in this section.

**Lemma 3.5.** *Let q be a query in the system, let o be an active data object, let $t_o^D$ be a point in time when o will be dropped from the q's window. If o is in q's k-skyband at $t_o^D$, it is also among q's top-k objects at $t_o^D$. More formally,*

$$\forall o \in S : (o \in S_q(t_o^D)) \Rightarrow (|W_q(t_o^D) \overset{q}{\triangleright} o| < k_q). \tag{3.17}$$

*Proof.* All data objects within the window of $q$ at $t_o^D$ are more recent than $o$ and thus will be dropped later from the window. Therefore, from Definition 3.16 we have $|W_q(t_o'^D)) \overset{q}{\blacktriangleright} o| = |W_q(t_o'^D)) \overset{q}{\triangleright} o| < k_q$. □

In other words, this lemma states that the oldest data object from a k-skyband is always among the top-k query objects at a point in time just before it will be dropped from the query window. Otherwise, if it were not among top-k objects, it would have previously been dominated by $k$ or more than $k$ objects.

### 3.3.2 Strict Candidate Pruning Algorithm

The *Strict candidate pruning Algorithm* (SA) continuously maintains a k-skyband associated with a query. SA applies strict pruning of dominated data objects after each object arrival, and stores strictly non-dominated objects associated with a query according to Definition 3.16.

Each data object is associated with three attributes: 1) its score for the query, 2) its dominance counter (i.e. the number of objects in the k-skyband that dominate it), and 3) the sequence number of an object that will expel it from the query window as stated in Definition 3.8. We represent the $k$-skyband in memory using two singly-linked lists sorted by descending object scores (i.e. ascending ranks) as shown in Figure 3.4, where each object score is written in the square that represents it. The *top list* stores $k$ best ranked non-dominated objects, while the *candidate list* stores other non-dominated objects for the query. Objects in the candidate list can become top list objects when some of the top list objects become dropped from the query window. Additionally, we use a self-balancing binary tree[1] (*time tree*) in memory

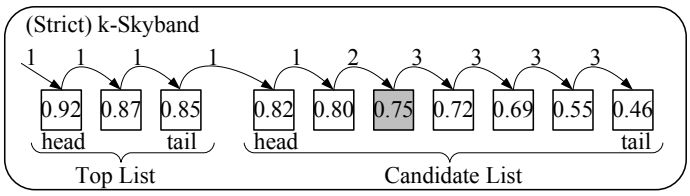



Figure 3.4: Adding an object *o* into a strict k-skyband.

to sort objects by ascending sequence numbers and efficiently locate dropped objects. Only top-k objects should be added to the time tree because, as shown by Lemma 3.5, other objects cannot be dropped from the query window.

The processing of an incoming data object is done in three steps as shown in Figure 3.4. We traverse elements in both lists starting from the head (highest ranked) of the top list to the tail (lowest ranked) of the candidate list. In the first step we find the number of object dominators for a new object by checking higher ranked objects, and increase the object dominance counter for every dominator we encounter[2] If we find the object is dominated by more than *k* k-skyband objects, we abort the procedure since the object is neither a top-k nor candidate object. Otherwise, we insert the object to its position in one of the lists in the second step and, in the third step, traverse objects with lower ranks and increase the dominance counter of each object dominated by the new object. The pseudo-code of the algorithm for processing of an incoming object is shown in Algorithm 1. The algorithm for processing incoming objects which are added to top list and reported as top-k objects (line 5) is presented in lines 2-18, while the algorithm for processing other objects is presented in lines 20-29. If we find that the dominance counter of a lower ranked object is larger than parameter *k* (lines 17 and 28), we will remove it from the k-skyband (lines 18 and 29). Additionally, if an incoming object is added to the top list, so the size of the list becomes larger than *k* (line 9), we will move the top list tail to the candidate list (lines 10-15).

After each new object arrival, SA checks the oldest object in the time tree to see if it is expelled (i.e dropped) from the query window by a newly arriving object. When this happens, the expelled object is removed from the time tree and k-skyband. The removal from the k-skyband can be easily handled because all other objects in it are more recent, and this object cannot dominate them. As shown in Lemma 3.5, the expelled object is always a top-k object and therefore we have to remove it from the top list. The algorithm for the removal of a dropped object from the k-skyband is shown in Algorithm 2. When a dropped object is located [3] (line 1) and removed (line 2), the head of the candidate list has to be moved to the top list to fill the empty slot (line 5). If the moved object is neither expired nor previously inserted

[1]To be more precise, we use a red-black tree that has a reference to the oldest object in the tree, such that the access to this object has amortized time complexity $O(1)$. A Fibonacci heap gave us very similar, but generally a slightly worse processing performance. For additional information about the data structures please check [64].

[2]Please note that although it is impossible for an incoming object to have dominators in the k-skyband at the time of its appearance since it is the youngest in the k-skyband, SA will check the domination property. This makes the algorithm generally applicable for the buffering approach we introduce in Section 3.5.

[3]In the programming languages with the low-level of abstraction it is possible to access a list element directly and delete it without traversing preceding objects. Since in the average-case the most of k-skyband objects will become dominated before dropped from the window, this traversing has a minor influence on the processing performance.

**Algorithm 1** SA: Process an incoming object $o$

**Require:** $u$, $\triangleright$, $o$, *toplist*, *candlist*, *timetree*, $k$
1: **if** $o \triangleright$ *toplist.tail* **then**
2: //step 1
3: count dominators of $o$ among its predecessors in *toplist*;
4: //step 2
5: report $o$ as a top-k object;
6: add $o$ to *toplist* and *timetree*;
7: //step 3
8: increase counters of dominated successors of $o$ in *toplist*;
9: **if** *toplist.size* $\geq k$ **then**
10: $t \leftarrow$ *toplist.tail*;
11: **if** $t$ is not dominated **then**
12: move $t$ from *toplist* to *candlist*;
13: remove $t$ from *timetree*;
14: **else**
15: remove it from *candlist* and *timetree*;
16: increase counters of dominated successors of $o$ in *candlist*;
17: **if** any *sucessor.counter* $\geq$ parameter $k$ **then**
18: remove *succesor* from *candlist*;
19: **else**
20: //step 1
21: count dominators of $o$ among its predecessors in both lists;
22: **if** *o.counter* $\geq k$ **then**
23: abort because $o$ is dominated;
24: //step 2
25: add $o$ to *q.candlist*;
26: //step 3
27: increase counters of dominated successors of $o$ in *candlist*;
28: **if** any *sucessor.counter* $\geq k$ **then**
29: remove *sucessor* from *candlist*;

in the query result stream (line 7), it will be instantly reported to the query (line 8).

### 3.3.3 Relaxed Candidate Pruning Algorithm

In practice, the continuous maintenance of a strict k-skyband produces a large processing overhead due to immediate pruning of dominated objects. Therefore, our original *Relaxed candidate pruning Algorithm* (RA) allows referencing of additional dominated data objects in memory, i.e. in the relaxed k-skyband. Opposed to SA, RA periodically prunes dominated objects from the relaxed k-skyband.

We represent the relaxed k-skyband in memory using two self-balancing binary trees[4]sorted by ascending ranks as shown in Figure 3.5, where the score of each object is written in the square that represents it. Each object is associated with two attributes: 1) its score for the query and 2) the sequence number of an object that will expel it from the query window. The *top tree* stores $k$ best ranked data objects, while

**Algorithm 2** SA: Remove a dropped object *o*

**Require:** *o*, *toplist*, *candlist*, *timetree*
1: traverse predecessors of *o* in *toplist*;
2: remove *o* from *toplist*;
3: **if** *candlist* is not empty **then**
4: $h \leftarrow$ *candlist.head*;
5: move *h* from *candlist* to *toplist*;
6: add *h* to *timetree*;
7: **if** it *h* neither expired nor previously reported **then**
8: report *h* as a top-k object;

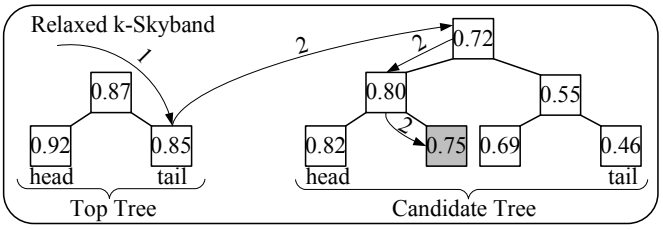



Figure 3.5: Adding an object *o* into a relaxed k-skyband.

the *candidate tree* stores other referenced objects for the query. Two attributes are associated with the query: 1) *pruning coefficient* $\gamma$, and 2) *candidate tree limit*. The value of the pruning coefficient can be chosen arbitrary. This coefficient represents the percent increase of the size of the candidate tree after a pruning that triggers the subsequent pruning. After each pruning, we set the candidate tree limit as $(1 + \gamma)$ multiplied by the new candidate tree size. The initial value of the candidate tree size should be a few times larger than *k*. Analogous to SA, we use a *time tree* to sort objects by ascending sequence numbers and efficiently locate dropped objects that need to be removed from the relaxed k-skyband.

The processing of an incoming data object is done in three steps, as shown in Figure 3.5. We first compare the incoming object score with the score of the object with the lowest rank in the top tree, i.e. the top tree tail. In the second step, the object is inserted either to the top tree if its score is higher than the top tree tail, or to the candidate tree. In the third step, the process of pruning dominated objects is triggered if the number of objects in the candidate tree is larger than the predefined limit. The pseudo-code of the algorithm for processing an incoming object is shown in Algorithm 3. If the object is added to the top tree (line 5) such that its size becomes larger than *k* (line 6), we move the top tree tail to the candidate tree (line 8). After each pruning (line 15), the new candidate tree limit is calculated (line 16).

To efficiently prune dominated objects from the candidate tree, we use an auxiliary self-balancing binary tree of size *k* called *dominator tree* to store top-k objects sorted by ascending times of object arrival. The pruning is done in two steps, as shown in Figure 3.6. We first fill the dominator tree with top tree objects. Then, in the second step, RA is traversing the candidate tree from the head to tail (i.e.

[4]To be more precise, we use two red-black trees. The top tree has a reference to its lowest ranked object (i.e. its tail), while the candidate tree has a reference to its head object, such that the access to these objects has amortized time complexity $O(1)$. Again, a Fibonacci heap gave us very similar, but generally a slightly worse processing performance.

**Algorithm 3** RA: Process an incoming object $o$

**Require:** $u$, $\triangleright$, $o$, *toptree*, *candtree*, *timetree*, $k$, *limit*, $\gamma$
1: //step 1
2: **if** $o \triangleright$ *toptree.tail* **then**
3: //step 2
4: report $o$ as a top-k object;
5: add $o$ to *toptree* and *timetree*;
6: **if** *toptree.size* $\geq k$ **then**
7: $t \leftarrow$ *toptree.tail*;
8: move $t$ from *toptree* to *candtree*;
9: remove $t$ from *timetree*;
10: **else**
11: //step 2
12: add $o$ to *candtree*;
13: //step 3
14: **if** *candtree.size* $\geq$ *limit* **then**
15: prune dominated objects from *candtree*;
16: set *limit* $= (1+\gamma) \cdot$ *candtree.size*;

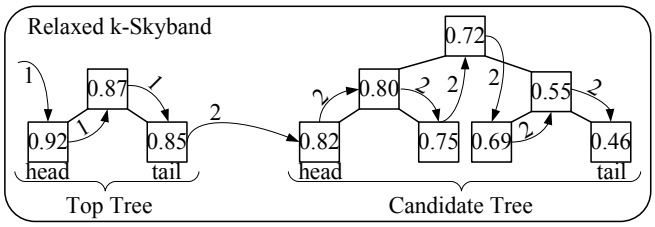



Figure 3.6: Pruning dominated objects from an RA candidate tree.

in descending scores order), while at the same time keeping $k$ most recent of already encountered objects in the dominator tree. Additionally, when it detects that an encountered object has appeared before the oldest element in the dominator tree (i.e. the dominator tree head), such an object is obviously dominated by $k$ objects in the dominator tree and is thus removed from the relaxed k-skyband. The pseudo-code of the algorithm for pruning dominated objects is shown in Algorithm 4. After each pruning, only non-dominated data objects, i.e. k-skyband objects stay in the relaxed k-skyband.

Analogous to SA, RA checks the oldest object in the time tree upon each new arrival to see if it is dropped from the query window. When this happens, the dropped object is removed from the time tree and relaxed k-skyband. The algorithm for removing dropped objects from the relaxed k-skyband is presented in Algorithm 5. As we add only top-k objects to the time tree, a removed object is always a top-k object and therefore we have to remove it from the top tree (line 1). However, some candidate tree objects (which are not added to the time tree) may also be dropped from the window, and these objects will be removed from the relaxed k-skyband either upon the next pruning or while filling the empty slot (lines 2-6). If the candidate tree contains an object that is still within the window (line 7) it will be moved to the top list to fill the empty slot (line 8). Additionally, if the moved object is neither expired nor previously

**Algorithm 4** RA: Prune dominated objects

**Require:** *toptree*, *candtree*, *timetree*
1: *domtree* $\leftarrow \{\emptyset\}$
2: **for all** *o* in *toptree* **do**
3: add *o* to *domtree*;
4: **for all** *o* in *candtree* **do**
5: add *o* to *domtree*;
6: $h \leftarrow$ *domtree.head*;
7: remove *h* from *domtree*;
8: **if** $h = o$ **then**
9: remove *o* from *candtree*;

**Algorithm 5** RA: Remove a dropped object *o*

**Require:** *o*, *toptree*, *candtree*, *timetree*,
1: remove *o* from *toptree*;
2: **repeat**
3: $h \leftarrow$ *candtree.head*;
4: **if** *h* is **not** null **then**
5: remove *h* from *candtree*;
6: **until** *h* is null **or** *h* is within the window;
7: **if** *h* is **not** null **then**
8: add *h* to *toptree* and *timetree*;
9: **if** *h* is neither expired nor previously reported **then**
10: report *h* as a top-k object;

inserted in the query result stream (line 9), it will be instantly reported as a top-k object (line 10).

## 3.4 Probabilistic Candidate Pruning Algorithm

In practice, most of the objects from a query k-skyband will never become top-k objects for the query and will never be reported as its answers. Therefore, in this section we present our original *Probabilistic candidate pruning Algorithm* (PA) which prunes k-skyband objects that have a small probability to become top-k objects in the future and maintains a significantly smaller set of candidate data objects. For this reason, PA is more efficient for top-k/w stream processing than deterministic algorithms since memory consumption is the most important efficiency measure for all streaming algorithms [114]. The PA we present in this section is an updated version of our original algorithm for top-k/w processing of random-order data streams published in [172]. The most important characteristics of a *random-order data stream model* is that any permutation of streaming data objects is equally likely to appear in a stream. This does not imply that object scores are random for a query, but that data objects are drawn independently from a non-time-varying distribution which is characteristic for a number of independent data sources, e.g. RSS feeds or large sensor networks. The random-order data stream model was originally introduced in [150], which is one of the first papers in the field of data stream processing with limited memory, and

has been used to describe and analyze a number of real-world application scenarios [101, 50, 51, 114]. In comparison to SA and RA, the main drawback of PA is that, as a probabilistic algorithm, it produces approximate results.

In this section we first present the mathematical background of PA which allows us to calculate an object's probability to become a top-k object in future, and second, we define the PA. As for the algorithms in the previous section, we present PA with respect to a single top-k/w query $q = \{u, \triangleright, k, n, t^A, t^C\}\}$ if otherwise not explicitly stated.

### 3.4.1 Mathematical Background

To explain the mathematical background of PA, let us assume the stream processor has unbounded memory which, at every point in time, can store all objects from the current query window.

**Definition 3.17** (Score List)**.** Let $q$ be a continuous top-k/w query, and let $t$ be a point in time when $q$ is active. We define the score list $L(t)$ of $q$ at $t$ as a list of data objects within the window of $q$ at $t$ which are ordered by their ascending ranks $L(t) \stackrel{\text{def}}{=} W_q(t)$.

Let us assume that at a point in time $t_o^A$ a new object $o$ arrives such that its initial rank is $l$ in the score list. The rank of this object will change during time as new objects arrive while the old ones are dropped from the window. Our goal is to find the probability that $o$ will become a top-k object in future. The initial rank $l$ of $o$ may have the following properties:

1. Top-k rank: $o$ is a top-k object at $t_o^A$,
2. Good rank: $o$ is not a top-k object at $t_o^A$ but has probability $p \geq \sigma$ to later become a top-k object, and
3. Poor rank: $o$ is not a top-k object at $t_o^A$ and has low probability $p < \sigma$ to later become a top-k object.

If $l$ is a top-k or good rank, $o$ is referenced in memory and maintained in the *probabilistic k-skyband*, or is ignored otherwise. To simplify the discussion, we assume the following: 1) we know only the scores (i.e. ranks) of objects in $L$, and do not know their actual sequence number in the data stream and 2) the cardinality of $L$ is $n$ at all points in time, i.e. we will ignore the initial period after query activation when $L$ is not full. Let a new rank of $o$ in $L(t')$ be $l' = l - x + y$ at a later point in time $t' > t_o^A$ where $x$ is the number of objects from $L(t_o^A)$ with a higher rank than $o$ at $t_o^A$ that have been dropped from the window during the time interval $(t_o^A, t')$, and $y$ is the number of objects that have arrived during $(t_o^A, t')$ and have a better rank than $o$. Hereafter we introduce two lemmas for calculating the probability related to objects with higher rank than $o$.

**Lemma 3.6.** *For data objects from a random-order stream, the probability that x of d objects are dropped from the window in the interval $(t_o^A, t')$ that have a higher rank than o is the following:*

$$p_d(l, n, d, x) = \frac{\binom{l-1}{x}\binom{n-l}{d-x}}{\binom{n-1}{d}}, \tag{3.18}$$

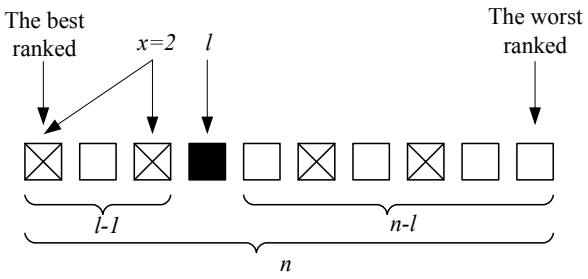



Figure 3.7: An example score list at a point in time when an object $o$ appears.

*where* $l \leq n$, $x \leq l-1$, $x \leq d$ *and* $d \leq n-1$.

*Proof.* We have to find the probability that among $d$ dropped objects, $x$ have higher ranks than $o$. Figure 3.7 depicts an example score list where $o$ is at position $l$ at $t_o^A$ (marked as a black square), while objects being dropped during $(t_o^A, t')$ are marked as checked squares. Every object in $L(t_o^A)$ except $o$, has an equal probability to be dropped from the window due to our previous assumption. Since they have arrived in $L$ before $o$, they will also be dropped from $L$ before $o$. Therefore, there are $\binom{n-1}{d}$ different possibilities to choose $d$ dropped objects from $n-1$ potentially dropped. Additionally, $x$ of $d$ dropped objects are ranked higher than $o$, and thus there are $\binom{l-1}{x}$ different possibilities to choose $x$ dropped objects from $l-1$ potentially dropped objects that are ranked higher than $o$, and $\binom{n-l}{d-x}$ different possibilities to choose $d-x$ from $n-l$ potentially dropped objects that are ranked lower than $o$. □

**Lemma 3.7.** *For data objects from a random-order stream, the probability that* $y$ *out of a objects have arrived during the interval* $(t_o^A, t')$ *that have a higher rank than* $o$ *is the following*[5]

$$p_a(l,n,a,y) = \frac{n}{n+a} \cdot \frac{\binom{l-1+y}{y}\binom{n-l+a-y}{a-y}}{\binom{n+a-1}{a}}. \tag{3.19}$$

*where* $l \leq n$ *and* $y \leq a$.

*Proof.* We have to find the probability that among $a$ arrived objects, $y$ are ranked higher than $o$. This is a conditional probability $p(E_1|E_2) = p(E_1 \cap E_2)/p(E_2)$, where event $E_2$ is "the rank of $o$ in $L(t_o^A)$ is $l$", and where event $E_1$ is "$y$ of $a$ arrived objects are ranked higher than $o$. The probability of event $E_2$ is $1/n$ because there are $n$ possible ranks of $o$ in $L(t_o^A)$. $p(E_1 \cap E_2)$ is the probability that the rank of $o$ in $L(t_o^A)$ is $l$ and that among $a$ arrived objects $y$ are ranked higher than $o$. Let us take a look at Figure 3.8, where an example score list is shown at a point in time $t'$. Object $o$ is shown as a black square, objects that have arrived before $o$ are shown as empty or checked squares, and objects that have arrived after $o$ are shown as gray squares. Please note that objects represented by gray squares have previously expired

[5] An important remark has to be made: The probability $p_a(l,n,a,y)$ is not equal to $\binom{a}{y} \cdot 2^{-a}$ because it would mean that the scores of data objects are totally random, which is usually not true for random-orders streams, where any permutation of streaming data objects is equally likely to appear. Additionally, the probability $p_a(l,n,a,y)$ is not equal to $\frac{1}{a+1}$ because it would mean that only permutations of streaming objects appeared after $o$ are equally likely to appear, but not the permutations of all streaming objects.

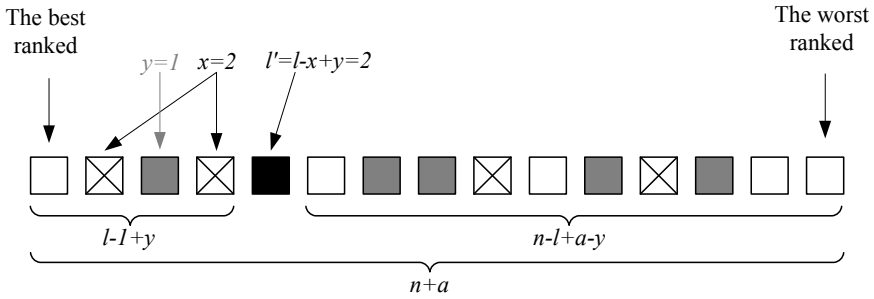



Figure 3.8: An example score list after several object appearances.

and thus are not elements of $L(t')$. We have to take these objects into account because our assumption is that any permutation of streaming data objects is equally likely in the stream. There are $\binom{n+a-1}{a}$ different possibilities to choose $a$ arrived objects from $n+a-1$ equally possible ranks in $L(t')$. Additionally, we know that the rank of $o$ in $L(t')$ is $l' = l - x + y$. Therefore, there are $\binom{l-1+y}{y}$ different possibilities to choose $y$ arrived objects from $l-1+y$ objects that are ranked higher than $o$. Analogously, there are $\binom{n-l+a-1}{a-y}$ different possibilities to choose $a-y$ arrived objects from $n-l+a-y$ objects that are ranked lower than $o$. Thereby, the probability $p(E_1 \cap E_2)$ is given as $(\binom{l-1+y}{y}\binom{n-l+a-y}{a-y})/(\binom{n+a-1}{a}(n+a))$, and then we have $p(E_1|E_2) = p(E_1 \cap E_2)/p(E_2) = n \cdot (\binom{l-1+y}{y}\binom{n-l+a-y}{a-y})/(\binom{n+a-1}{a}(n+a))$. □

At each processing step, a new object arrives in the system, and according to expression (3.4) it replaces an object within the window that has the oldest time of arrival in $L$. The probability that object $o$ has rank $l'$ at step $i$ is the following

$$p_s(l',l,n,i) = \begin{cases} \sum_{j=|r|}^{\min(l-1,i)} p_d(l,n,i,j) \cdot p_a(l,n,i,j-|r|) & \text{if } l' \leq l, \\ \sum_{j=|r|}^{\min(l'-1,i)} p_d(l,n,i,j-|r|) \cdot p_a(l,n,i,j) & \text{if } l' > l, \end{cases} \tag{3.20}$$

where $r = l' - l$ is a relative rank of $o$ in step $i$. Please note, that during the last processing step, i.e. when $i = n-1$, just before $o$ is dropped from the window, all other objects that have arrived before $o$ are already dropped. Therefore, using expression (3.19) we can calculate the probability that object $o$ has rank $l'$ at step $i = n-1$ as

$$p_s(l',l,n,n-1) = p_a(l,n,n-1,l'-1) = \frac{n}{2n-1} \cdot \frac{\binom{l+l'-2}{l'-1}\binom{2n-l-l'}{n-l'}}{\binom{2n-2}{n-1}} = \frac{n}{2n-1} \cdot \frac{\binom{n-1}{l'-1}\binom{n-1}{l-1}}{\binom{2n-2}{l+l'}}. \tag{3.21}$$

In our paper [172], we have used the probability $\sum_{l'=1}^{k} p_s(l',l,n,n-1)$ as a criterion for the detection of good ranks. In the next theorem we further tune this criterion to achieve a better approximation of the probability that an object will become a top-k object before it is dropped from the query window.

**Theorem 3.8.** *Let q be a top-k/w query over a count-based window of size n, let o be a data object with*

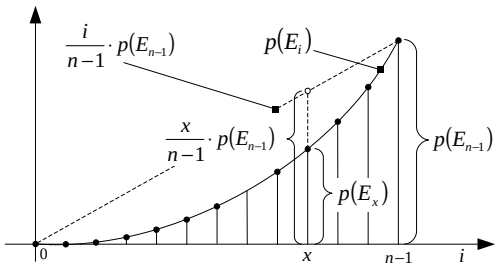



Figure 3.9: Approximation of the probability that an object is a top-k/w object at a processing step.

*rank $l > k$ within the window of $q$ at a point in time $t_o^A$. For data objects from a random-order data stream, an upper bound of the probability $p_t(l,n,k)$ that $o$ will become a top-k object before is $o$ is dropped from the window of $q$ is the following:*

$$p_b(l,n,k) = \frac{n^2}{4n-2} \cdot \sum_{l'=1}^{k} \frac{\binom{n-1}{l'-1}\binom{n-1}{l-1}}{\binom{2n-2}{l+l'}}. \tag{3.22}$$

*where $l \leq n$ and $l' \leq n$.*

*Proof.* For each step $i = 1, 2, \ldots, n-1$, let $E_i$ be the event "$o$ is a top-k object at step $i$". Using expression (3.20) we calculate the probability of an event $E_i$ as $p(E_i) = \sum_{l'=1}^{k} p_s(l', l, n, i)$. Let $p_t(l,n,k) = p(\sum_{i=0}^{n-1} E_i)$ be the probability of $o$ being a top-k object at any step $i$. Obviously, events $E_0, E_1, \ldots, E_{n-1}$ are dependent, and thus we can use inequality $p(\sum_{i=0}^{n-1} E_i) \leq \sum_{i=0}^{n-1} p(E_i)$ to bound the value of $p_t(l,n,k)$. For large values of $n$, it is difficult to exactly calculate this bound as it has $n-1$ summands. Our calculations show that $p(E_i)$, as a discrete function of step $i$, is monotonically increasing and convex for all starting ranks for which $\sum_{i=1}^{n-1} p(E_i) < 1$. Since $o$ is not a top-k object upon its arrival, for all such starting ranks we have $p(E_0) = 0$. Then, using the triangle shown in Figure 3.9, we bound the value of each $p(E_i)$ from above using its projection $\frac{i}{n-1} \cdot p(E_{n-1})$ on the triangle diagonal: $\sum_{i=0}^{n-1} p(E_i) \leq \frac{n}{2} \cdot p(E_{n-1})$. Finally, using $p(E_{n-1}) = \sum_{l'=1}^{k} p_s(l', l, n, n-1)$ and expression (3.21) we conclude that the following property holds: $p_t(l,n,k) \leq \sum_{i=0}^{n-1} p(E_i) \leq \frac{n}{2} \cdot p(E_{n-1}) = p_b(l,n,k) = \frac{n^2}{4n-2} \cdot \sum_{l'=1}^{k} [\binom{n-1}{l'-1}\binom{n-1}{l-1} / \binom{2n-2}{l+l'}]$. □

The previous theorem defines the probabilistic criterion we use upon a new object arrival to decide whether its initial rank for a query is good or poor, and thus whether it needs to be maintained in memory or not. Poor ranks are those with an upper bound $p_b(l,n,k) < \sigma$, while the following condition holds for good ranks: $p_b(l,n,k) \geq \sigma$, where $\sigma$ is the predefined probability of error. For data stream processing scenarios with a specific $\sigma$, we can compute the number of objects to be maintained in the probabilistic k-skyband using the probabilistic criterion defined in expression (3.22). Let us find an integer $k+limit$ which is the *last good rank* of an object within the probabilistic k-skyband such that $p_b(k+limit, n, k) \geq \sigma$ and $p_b(k+limit+1, n, k) < \sigma$. The value of $k+limit$ has to be calculated numerically and here we sketch the algorithm for finding *limit*. The algorithm requires $\sigma$, $k$ and $n$ as input parameters and calculates $p_b(l,n,k)$ for $l = k, k+1, \ldots$ until we find the first rank $f$ such that $p_b(f,n,k) < \sigma$, and thus $k + limit = f - 1$ is the last good rank within the score list. We call $limit = f - 1 - k$ the *probabilistic limit* on the number

Table 3.2: Probabilistic limit on the number of candidate objects.

| $n \setminus k$ | 1 | 2 | 5 | 10 | 20 | 50 | 100 | 200 | 500 |
|---|---|---|---|---|---|---|---|---|---|
| $10^3$ | 17 | 20 | 25 | 31 | 39 | 54 | 70 | 88 | 103 |
| $2 \cdot 10^3$ | 18 | 21 | 26 | 32 | 41 | 57 | 75 | 99 | 135 |
| $5 \cdot 10^3$ | 20 | 22 | 28 | 34 | 43 | 61 | 80 | 107 | 157 |
| $10^4$ | 21 | 24 | 29 | 36 | 45 | 63 | 84 | 112 | 167 |
| $2 \cdot 10^4$ | 22 | 25 | 30 | 37 | 46 | 65 | 86 | 116 | 175 |
| $5 \cdot 10^4$ | 23 | 26 | 32 | 39 | 48 | 68 | 90 | 121 | 183 |
| $10^5$ | 24 | 27 | 33 | 40 | 50 | 70 | 92 | 124 | 188 |
| $2 \cdot 10^5$ | 25 | 28 | 34 | 41 | 51 | 72 | 95 | 128 | 193 |
| $5 \cdot 10^5$ | 26 | 29 | 36 | 43 | 53 | 74 | 98 | 132 | 199 |
| $10^6$ | 27 | 31 | 37 | 44 | 55 | 76 | 100 | 135 | 203 |

of object candidates that need to be maintained in the probabilistic k-skyband, where the probability of error is upper-bounded by $\sigma$. Table 3.2 lists the probabilistic limits for $\sigma = 0.001$ and some typical values of $k$ and $n$, and shows that the memory overhead due to maintenance of object candidates for increasing values of $n$ and $k$ decreases and becomes almost negligible for large values of $n$ and $k$ (e.g. $0,0203\%$ for $n = 10^6$ and $k = 500$).

### 3.4.2 Algorithm Description

PA maintains the probabilistic k-skyband in memory using two self-balancing binary trees[6] sorted by descending scores (i.e. by ascending ranks) as shown in Figure 3.10, where the score of each object is written within the square that represents it. Therefore, the probabilistic k-skyband structure is similar to the relaxed k-skyband, and each object is associated with:processing- 1) its score and 2) the sequence number of an object that will expel it from the query window. The *top tree* stores $k$ best ranked data objects, while the *candidate tree* stores other referenced objects for a query. Two attributes are associated with each query: 1) *probabilistic limit* on the number of candidates, and 2) *threshold*. A probabilistic limit is calculated upon query activation using parameters $\sigma$, $k$ and $n$, while threshold is initially set to the lowest possible score value. When the number of objects in the candidate tree equal the probabilistic limit, the threshold value is set to the score of the candidate tree tail object. Analogous to SA and RA, we use a *time tree* to sort objects by ascending sequence numbers to efficiently locate objects that need to be removed from the probabilistic k-skyband.

The processing of an incoming data object is done in four steps, as shown in Figure 3.10. We first compare the object score with the threshold and abort the procedure when it is lower that the threshold because the object rank is poor, or continue with the second step otherwise. In the second step we additionally compare the object score with the score of the top tree tail. In the third step, the object is inserted either to the top tree if its score is higher than the top tree tail, or to the candidate tree. The candidate tree tail is removed from the probabilistic k-skyband if the number of objects in the candidate tree is larger than the probabilistic limit, and subsequently the threshold is updated. The pseudo-code of the algorithm for processing an incoming object is shown in Algorithm 6. If the object is added to the top tree (line

[6]These trees are identical to the top and candidate trees of RA.

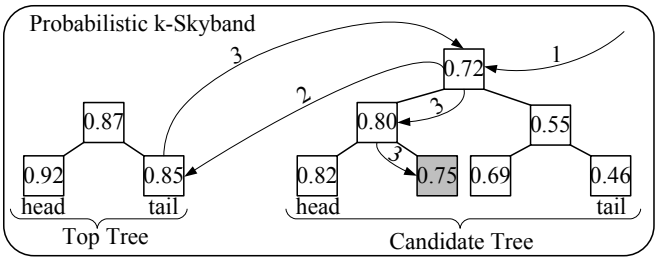



Figure 3.10: Adding an object $o$ to a probabilistic k-skyband.

**Algorithm 6** PA: Process an incoming object $o$

**Require:** $u$, $\rhd$, $o$, *toptree*, *candtree*, *timetree*, $k$, *limit*, *threshold*

```
1:  //step 1
2:  if o ◁ threshold then
3:      abort because o has a poor rank
4:  else
5:      //step 2
6:      if o ▷ toptree.tail then
7:          //step 3
8:          report o as a top-k object;
9:          add o to toptree;
10:         if toptree.size ≥ k then
11:             move toptree.tail from toptree to candtree;
12:     else
13:         //step 3
14:         add o to q.candtree;
15: add o to timetree;
16: //step 4
17: if candtree ≥ limit then
18:     remove candtree.tail from candtree and timetree;
19:     set threshold = candtree.tail.score; //new candtree.tail
```

9) such that its size becomes larger than $k$ (line 10), we move the top tree tail to the candidate tree (line 11). Please note that in contrast to SA and RA, PA adds both top tree and candidate tree objects to the time tree (line 15). This is necessary because candidate tree can comprise dominated objects which can be dropped from the query window.

Analogous to SA and RA, PA checks the oldest object in the time tree upon each new arrival to find out whether this object is dropped from the query window. When this happens, the dropped object is removed from the probabilistic k-skyband. The algorithm for removing dropped objects is presented in Algorithm 7. Since we add all probabilistic k-skyband objects to the time tree, a dropped object is either a top-k object or a candidate tree object (line 1). In the latter case we just remove it from the trees (line 9). In the former case (lines 2-7), we remove it from the trees (line 2), and move the candidate tree head to the top tree to fill the empty slot in top tree (line 5). If the moved object is neither expired nor previously inserted in the query result stream (line 6), it will be instantly reported as a top-k object (line 7). After

**Algorithm 7** PA: Remove a dropped object $o$

**Require:** $u$, $\triangleright$, $o$, *toptree*, *candtree*, *timetree*, *threshold*
1: **if** $o \triangleright$ *toptree.tail* **then**
2: remove $o$ from *toptree* and *timetree*;
3: **if** *candtree* is not empty **then**
4: $h \leftarrow$ *candtree.head*;
5: move $h$ from *candtree* to *toptree*;
6: **if** $h$ is neither expired nor previously reported **then**
7: report $h$ as a top-k object;
8: **else**
9: remove $o$ from *candtree* and *timetree*;
10: approximate *threshold*;

each object removal, we have to approximate the threshold value (line 10) to avoid resetting it to 0. The threshold is approximated so that its previous value is decreased by an absolute difference in scores of two lowest ranked objects from the candidate tree. When the candidate tree becomes full again, the score of its tail will become the new threshold value (lines 17 and 19 in Algorithm 6).

## 3.5 Enhancement of Deterministic Algorithms with Query Filters

In practice, SA and RA may insert many data objects with low ranks into the query k-skyband, although such objects will soon become dominated by higher ranked and more recent objects. Such objects are not dominated when they arrive into the system due to their late arrivals and would normally be stored in the query k-skyband, although their potential to become top-k objects is rather low. In this section we outline enhanced versions of SA and RA which improve algorithm performance by filtering out recent objects with low ranks and delaying their insertion into the k-skyband.

To enable filtering of recent objects, we employ a special FIFO buffer. The buffer stores data objects from the query window of size $b$ according to Definition 3.8, where $n$ is replaced by $b$, and we denote the set of data objects in the buffer by $B$.

**Definition 3.18.** *(Data Objects in the Recent Buffer)* Let $t$ be a point in time, let $b \in \mathbb{N}$, and let $o_z$ be the last object that has appeared in the system at such that $\nexists o \in S : (t > t_o^A > t_{o_z}^A)$. We define a set $B(t) \subset S$ of data objects in the recent buffer at $t$ as follows:

$$B(t) \stackrel{\text{def}}{=} \{o_i : (o_i \in S) \wedge (t > t_{o_i}^A) \wedge (i > z - b)\}, \tag{3.23}$$

where $i$ and $z$ are sequence numbers of $o_i$ and $o_z$, respectively.

Typically $b \ll n$, and the buffer is represented in memory as a singly-linked list of objects. To efficiently process data objects from the buffer, we associate an auxiliary k-skyband to a query—*query filter*—filled with data objects from the buffer. The query filter enables filtering of objects with low ranks by avoiding their adding to the query k-skyband at the time of arrival: The object will rather be

temporary maintained in the query filter while it is within the buffer, and the query filter will make the second attempt to insert the object into the query k-skyband, just before it is dropped from the buffer. At the later point, such an object will probably be dominated by younger objects with higher ranks, and will be discarded, unless the quality of the data stream with respect to the query changes and newly arriving objects have mainly lower ranks than the original object.

The next theorem explains when and under which circumstances data objects can be delayed in the recent buffer.

**Theorem 3.9.** *Let $q$ be a query in the system, let $B$ be a recent buffer of size $b$, let $t^A_{o_i}$ be a point in time when $q$ is active and when a new data object $o_i$ arrives in the system that is not a top-k data object from the buffer at $t^A_{o_i}$ , and let $t^{E'}_{o_i} > t^A_{o_i}$ be a point in time when $o_i$ is expelled from the buffer. The data object $o_i$ cannot become a top-k object of $q$ while in the buffer (i.e. at any point in time $t$ where $t^{E'}_{o_i} \geq t > t^A_{o_i}$) if the following condition holds*

$$b \leq \frac{n+1}{2}. \tag{3.24}$$

*Proof.* If every element from $B(t^A_{o_i})$ is still within the window of $q$ at $t^{E'}_{o_i}$, the following holds $\forall t : (t^{E'}_{o_i} \geq t > t^A_{o_i}) \Rightarrow (|O \subseteq (B(t) \overset{q}{\rhd} o) \subseteq (W_q(t) \overset{q}{\rhd} o)| = k_q)$, and then from expression (3.6) we will have $\nexists t : (t^{E'}_{o_i} \geq t > t^A_{o_i}) \wedge (o \in T_q(t))$. Let $o_j \in B(t^A_{o_i})$ be the least recent object in $B$ which will be expelled from $B$ upon the next arrival (i.e. $i = j + b - 1$). If $o_j$ is still within the window of $q$ at $t^{E'}_{o_i}$ all other object in the buffer at $t^{E'}_{o_i}$ are more recent than $o_j$. From expression (3.4) we have $j + n - 1 \geq i + b - 1$, and then $n \geq b + i - j = 2 \cdot b - 1$. □

In other words, if the previous condition holds, the objects that are delayed in the buffer cannot become top-k query objects while they are in the buffer. In practice, buffer size is typically much smaller than the query window size and thus the condition holds. Furthermore, when an object is not a top-k buffer object just before it will be expelled from the buffer, this object is dominated by $k$ buffer objects, i.e. it is dominated for the query, and can safely be discarded. Otherwise, if the object is a top-k buffer object just before it is dropped from the buffer, it should be added into the query k-skyband as it has the potential to become a top-k/w object.

The query filter may be implemented as any of the previously defined k-skybands, i.e. either strict, relaxed or probabilistic k-skyband. Therefore three types of filters are possible: *strict filter* (SF), *relaxed filter* (RF) and *probabilistic filter* (PF). Note that the query filter is quite similar to the top-k/w query because both use the same scoring function, score comparator and parameter $k$, however, the major difference is the window size—$b$ for the query filter, and $n$ for the query. Thus the k-skyband structure associated with a query filter is the same as for a top-k/w query requiring that only buffer objects are possible filter k-skyband elements: top-k objects from the buffer are maintained in the filter top list/tree, while others are within the filter candidate list/tree.

The processing algorithm works as follows: Each incoming data object is first added to the recent buffer and then processed by the query filter. The query filter will insert an incoming object into the query k-skyband only if it is a top-k object in the filter at the time of its arrival, or will otherwise try to insert

**Algorithm 8** Filter: The 2nd insertion attempt of an object $o$

**Require:** $u, \triangleright, o, filter$
1: **if** $o \triangleright filter.topstructure.tail$ **then**
2: **if** $o$ is not inserted in the first insertion attempt **then**
3: process $o$ for the query;
4: **if** $o$ is stored in the *filter* **then**
5: remove $o$ from *filter*;

it in the second attempt[7](during the removal from the buffer) as defined in Algorithm 8. In the second insertion attempt we will process $o$ for the query (line 3) and remove it from the filter (line 5) if it is still referenced there (line 4) only if less than $k$ higher ranked objects are left in the filter (line 1) and it was not inserted in the first attempt (line 2). Otherwise, the object will just be removed from the filter (lines 4-5) and will not be processed for the query since it is dominated for the query, as we prove in the following theorem. Please note that the original versions of SA, RA and PA have to be slightly modified to be used for filter processing. We replace line 5 in Algorithm 1, line 4 in Algorithm 3 and line 8 in Algorithm 6 with "process $o$ by the query filter" instead of reporting $o$ as a top-k object, and additionally we delete lines 7-8 in Algorithm 2, lines 9-10 in Algorithm 5 and lines 6-7 in Algorithm 7.

**Theorem 3.10.** *Let $q$ be a query in the system, let $B$ be a recent buffer of size $b$, and let $t_{o_i}^{E'} = t_{o_j}^{A}$ be a point in time when $q$ is active and when a new data object $o_j$ appears in the system, enters and expels from the buffer a data object $o_i$ (i.e. $i = j - b$) that is valid for $q$ at $t_{o_i}^{E'}$. The data object $o_i$ cannot become a top-k object of $q$ at any future point in time $t > t_{o_i}^{E'}$ if at $t_{o_i}^{E'}$ it is ranked worse than $k_q$ objects left in buffer, more formally:*

$$\forall o_i \in S : [|B(t_{o_i}^{E'}) \overset{q}{\triangleright} o_i| \geq k_q] \Rightarrow (\nexists t > t_{o_i}^{E'} : o_i \in T_q(t)) \tag{3.25}$$

*Proof.* All data objects left in the recent buffer at $t_{o_i}^{E'}$ are more recent than $o_i$ and thus will be dropped later from the window of $q$. Thus we have $|B(t_{o_i}^{E'}) \overset{q}{\blacktriangleright} o_i| = |B(t_{o_i}^{E'}) \overset{q}{\triangleright} o_i| \geq k_q$, and then from the Theorem 3.1 directly follows that $o_i$ cannot become a top-k element of $q$ at any future point in time $t > t_{o_i}^{E'}$. □

Note that it is not necessary to have a deterministically correct set of top-$k$ objects from the buffer in a query filter to apply filters with SA and RA. Actually, any set of $k$ objects from the buffer can be used as a filter, and the deterministically correct set will prune the largest number of dominated objects. For this reason, we can use the probabilistic k-skyband as a filter, while keeping the exactness of query results and therefore the combination of SA and RA with PF is also a deterministic algorithms. We denote the algorithms that extend the original SA and RA with different versions of the query-filter as follows: SASF, SARF, SAPF, RASF, RARF and RAPF.

[7]Please note that during the processing in the second attempt, it is possible that some more recent objects are located in the query k-skyband, and therefore we must check if the currently processed object is dominated by them. This only refers to SA, whose presented version already supports this possibility, see line 3 of Algorithm 1.

## 3.6 Supporting Multiple Top-k/w Queries

This section considers a more general processing case when more then one top-k/w query is simultaneously active in the system. This case requires a few additional data structures and some modifications when compared to a single query:

1. We need a *query tree*, a self-balancing binary tree[8] for storing all active queries and query filters in the system. As filters forward data objects to their queries, they have references to their queries. Therefore, when using SA and RA with filters, the filters must have references to their queries, and then we put them, but not their queries, to the query tree.

2. To store all data objects that are referenced by at least one query or filter, we use a self-balancing binary tree[8] named *object tree*.

3. Sharing a *common time tree* among all queries and filters in the system is more efficient approach than having separate time trees for all objects because the later approach requires an additional time tree for sorting of all query time trees by their heads (i.e. least recent objects), which then produces additional processing due to tree synchronization.

4. We apply a *single object buffer* which is shared among all query filters in the system.

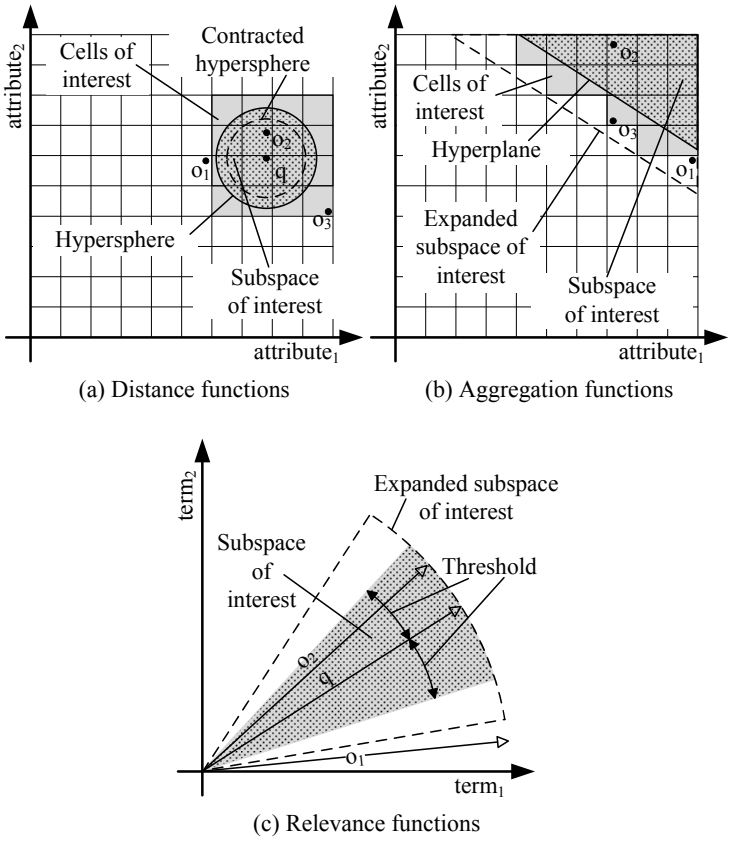
5. SA, RA and PA associate some attributes (e.g. score) to each object referenced by a query/filter. Since these attributes are query/filter specific, we create an object wrapper for each object referenced by a query/filter, and associate these attributes to the wrapper instead of the object itself. Each such wrapper has two additional attributes: a reference to its object in the object tree, and a reference to its query in the query tree.

In such a setting, the naive approach to process an incoming data object is to sequentially traverse the query tree and process this object for each traversed query/filter. Additionally, before each object processing we always remove dropped objects, i.e. object wrappers, from the common time tree and their query/filter k-skybands. Sequential processing is supported by all three algorithms, including versions with filters, but is usually inefficient. In the next subsection we explain how use query indexing techniques can be used to minimize the number of queries/filters that have to be informed about a new object arrival.

### 3.6.1 Indexing Non-Generic Top-k/w Queries

If we closely analyze PA, Algorithm 6 states that a query does not need to be informed about a newly arrived object if its score is worse than the query threshold. The presented versions of deterministic algorithms (SA and RA) work by processing all incoming data objects and do not define such a threshold. However, the versions of SA and RA with filters define a threshold as the score of the filter element with rank $k$, in the case of SF and RF, or as the filter threshold in the case of PF, because in both insertion attempts only objects with ranks higher than such a threshold are forwarded to the corresponding query. Both thresholds values change in time with new object arrivals and droppings.

[8] To be more precise, we again use a red-black tree. This tree allows fast adding and removing of queries (i.e. data objects).




(a) Distance functions (b) Aggregation functions

(c) Relevance functions

Figure 3.11: Indexing of top-k/w queries for various query scoring functions.

Query indexing reduces the number of data objects that a query/filter needs to process by skipping those objects that will certainly not be referenced. Our experimental evaluation shows that query indexing can significantly improve processing performance in certain situations. The query indexing techniques cannot be used for generic queries because they completely depend on data object representation and corresponding scoring functions. Hereafter we sketch indexing techniques for the three prominent categories of scoring functions: distance, aggregation, and relevance functions.

Distance and aggregation functions are applied to structured data where data objects are represented as points in a multidimensional attribute space. In accordance to existing approaches [141, 143, 32], we use a regular grid to index queries/filters in such attribute space. A regular grid divides an attribute space in cells of equal size, while a threshold defines the subspace of interest for a query/filter. In case of distance functions, each threshold represents the radius of the subspace of interest around a query/filter point, as depicted in Figure 3.11a. Similarly, for aggregation functions each threshold represents a hyperplane in the attribute space that defines the query/filter subspace of interest, as shown Figure 3.11b. Please note that the space of interest is covered by the cells of interest from the regular grid that encompass a larger portion of the attribute space than the subspace of interest itself. Obviously, a query/filter subspace is varying in time due to threshold changes. The subspace can either expand or contract as shown in Figures 3.11a and 3.11b. Query indexing changes the algorithms in the following: Upon each new object

arrival, we first find the grid cell in which our object resides, and then find a subset of queries/filters whose cells of interest encompass this cell to inform them about the appearance of the new object. For example, we will inform $q$ in Figure 3.11 about the appearance of object $o_2$, but not about object $o_1$ and $o_3$.

Relevance functions represent queries and data objects as vectors in a vector-space, and score, i.e. relevance, is defined as the cosine of the angle between a data object and query vectors. In this case, the query threshold represents the cosine of the angle between the query vector and the edge of the cone-like query subspace of interest, as shown in Figure 3.11c. Such a subspace also expands or contracts with new object arrivals and droppings. To our knowledge, the indexing of top-k/w queries in vector-space is an open research problem [142] and is not discussed further in this chapter.

## 3.7 Complexity Analysis

In this section we analyze time and space complexity of the previously defined algorithms SA, RA, and PA for both the average and worst-case scenarios. In the average-case scenario, we assume that any permutation of streaming data objects is equally likely to appear in a data stream. In the worst-case scenario, the scores of data objects are monotonically decreasing in time, and thus every incoming object has to be stored in memory as it will become a top-k object at a later point in time when $n-k$ older objects are dropped from the query window. Note that complexity analysis is performed with respect to a single top-k/w query $q = \{u, \rhd, k, n, t^A, t^C\}$, if otherwise not explicitly stated, for which the score comparator $\rhd$ is defined to assign higher ranks to objects with higher scores.

### 3.7.1 Space Complexity Analysis

We express space complexity as the number of data objects referenced in memory per query.

The space complexity of SA is $O(n)$ in the worst-case as all data objects will become top-k objects. In the average-case scenario, the expected number of data objects in a strict query $k$-skyband is $O(k \cdot ln(\frac{n}{k})) = O(n')$, as shown in [204]. Similarly, the worst-cased space complexity of RA is $O(n)$, and $O(n' + \frac{\gamma}{2} \cdot (n' - k)) = O(n' \cdot (1 + \frac{\gamma}{2}) - \frac{\gamma}{2} \cdot k)) = O(n' \cdot (1 + \frac{\gamma}{2})) = O(n')$ in the average-case because the number of data objects referenced by each query varies between $n'$ and $n' + \gamma \cdot (n' - k)$, where $\gamma$ is the pruning coefficient. The space complexity of PA is always $O(k + limit)$, where *limit* is the probabilistic limit on the number of object candidates. PA will potentially generate a large error in the worst-case scenario because it discards many incoming objects without knowing that they will become top-k objects in future. In the opposite case when scores of incoming data objects are monotonically increasing, PA will not generate error.

Please note that the space complexity for a query filter (i.e. SF, RF or PF) can be expressed analogously as for the corresponding query without filter (i.e. SA, RA or PA): we just need to replace $n$ with $b$ in the previous expressions, where $b$ is the number of buffer objects.

The worst-case space complexity of a query k-skyband for SA and RA with filters is $O[n-(b-k)] = O(n-b+k)$ due to the fact that $b-k$ objects from the recent buffer will be kept in the query filter. In the

average-case, only $k$ of $b'$ non-dominated objects in the buffer will be inserted to a query k-skyband. Thus the space complexity of a query k-skyband is $O(n'-b'+k) = O(k \cdot ln(\frac{n}{k}) - k \cdot ln(\frac{b}{k}) + k) = O(k \cdot (1 + ln(\frac{n}{b})))$ and $O(k \cdot (1 + ln(\frac{n}{b})) \cdot (1 + \frac{\gamma}{2}) - \frac{\gamma}{2} \cdot k) = O(k \cdot (1 + ln(\frac{n}{b})))$ for SA and RA with filters, respectively. It is important to recall that in case when query filters are used, there is an additional overhead of $b$ objects stored in the recent buffer. However, this buffer is shared by all queries in the system.

For the queries with filters, the query indexing does not affect the space complexity of a query k-skyband since it only restrains the processing of low-scored data objects by the associated query filter. Additionally, it also does not affect the space complexity of PA and PF since it only restrains the processing of objects which are scored worse than the threshold as shown in Algorithm 6. On the contrary, the indexing lowers the space complexity of SF and RF by avoiding the insertion of low-scored data objects.

### 3.7.2 Time Complexity Analysis

For the time complexity analysis, we assume that $m$ top-k/w queries with parameters $k$ and $n$ are processed simultaneously as they apply the common time tree which influences processing time. Furthermore, each incoming data object is processed sequentially, while in each processing cycle a new data object appears and the oldest is dropped from all query windows. Note that we express time complexity as the complexity of a processing cycle per query.

In the worst case scenario SA has time complexity $O(n + ld(m \cdot k))$, where $ld(m \cdot k)$ is due to maintenance of the common time tree, while it is $O(n')$ in the average case. The time complexity of RA in the worst-case is $O(ld(n-k)^2 + ld(m \cdot k))$, because the pruning process will never be started, while $ld(n-k)^2$ is due to addition and removal of objects from a candidate tree. The time complexity of RA in the average-case is $O(ld((n'-k) \cdot (1 + \frac{\gamma}{2}))^2) = O(ld(n'))$, which takes into account the addition of an incoming data object to a candidate tree, and object pruning every $\gamma \cdot (n'-k)$ processing cycles. For PA, in the worst-case, $k + limit$ data objects will be added at every $n$-th processing cycle into the probabilistic k-skyband, while on average $k + limit$ objects will be added per $n$ incoming objects, where $limit$ is the probabilistic limit on the number of candidates. As $k + limit \ll n$, the time complexity is $O(1 + 2 \cdot \frac{k+limit}{n} \cdot ld(m \cdot (k + limit))) \approx O(1)$ in both cases. The first term $\frac{k+limit}{n} \cdot ld(m \cdot (k + limit))$ is related to addition of an incoming object with a top or good rank to the time tree, while the second term relates to removal of an object with the worst rank in the candidate tree from the time tree. Note that the most time-consuming task per PA cycle is the object score calculation.

Note that processing with filters adds additional processing cycles and hereafter we analyze time complexity of the SA and RA with query filters. We express time complexity for a query filter maintenance by replacing $n$ with $b$ in the expressions for the SA, RA and PA without filters as follows: $O(b + ld(m \cdot k))$ and $O(k \cdot ln(\frac{b}{k})) = O(b')$ for SF, $O(ld(b-k)^2 + ld(m \cdot k))$ and $O(ld(b'))$ for RF, and $O(1)$ for PF. We get the worst-case time complexity of a query k-skyband maintenance by replacing $n$ with $n-b+k$ in the expressions for the SA and RA without filters: $O(n-b+k+ld(m \cdot (n-b+k)))$ and $O(ld(n-b)^2 + ld(m \cdot (n-b+k)))$ for the strict and relaxed k-skyband, respectively. In the average-case, the number of insertions into a query k-skyband is $n \cdot \frac{k}{b}$ per window size $n$. In other words an insertion happens every $\frac{k}{b}$-th cycle on average because $(n \cdot \frac{k}{b})/n = \frac{k}{b}$. From the space complexity analysis we

know that the average space complexity of a query k-skyband is $O(k \cdot (1 + ln(\frac{n}{b})))$ in this case. Therefore, the average-case time complexity of a query k-skyband maintenance for the SA and RA with filters is $O(\frac{k}{b} \cdot k \cdot (1 + ln(\frac{n}{b}))) = O(\frac{k^2}{b} \cdot (1 + ln(\frac{n}{b})))$ and $O(\frac{k}{b} \cdot ld(k \cdot (1 + ln(\frac{n}{b}))))$, respectively.

The indexing of queries improves the processing performances of query filters, but not of their query k-skybands due to the fact that the number of data objects processed by a query k-skyband is not affected by the indexing. On the contrary, the processing performance of a query filter is affected by the indexing such that it completely depends on the data distribution and indexing technique. We refer the reader interested in this thematic to the complexity analysis of regular grid operations in the case of distance and aggregation scoring functions [141, 143].

## 3.8 Experimental Evaluation

In this section we present an experimental study comparing the previously defined algorithms, namely, SA, RA, PA, SASF, SARF, SAPF, RASF, RARF, RAPF. All algorithms are implemented in Java and experiments were performed on a Linux based PC with 2.13Ghz Intel® Core™2 Duo CPU (with one core disabled) and 2GB of memory.

We have selected sliding window k-NN queries as a use case for our evaluation since k-NN queries are regarded as one of the most prominent top-k/w problems as we discuss further in Section 3.9. Both queries and data objects in our experiments are represented as points in a $d$-dimensional Euclidean space. The score of an object $o$ with respect to a query $q$ is calculated using the following formula: $u_q(o) = d(p_o, p_q) = [\sum_{i=1}^{d}(v_i - v_i)^2]^{\frac{1}{2}}$, where $p_o = \{v_1, v_2, \ldots, v_d\}$ and $p_q = \{v_1, v_2, \ldots, v_d\}$ are points representing the object $o$ and query $q$, respectively. The score comparator $\overset{q}{\triangleright}$ is defined such that lower object scores imply higher ranks.

For the sake of simplicity, we omit subscripts of top-k/w query parameters in this section, if otherwise not explicitly stated.

We have used one real and two synthetically generated datasets in the experimental evaluation. In particular, the real dataset used in experiments is the LUCE deployment data (environmental data collected from large-scale wireless sensor networks within the project SensorScope[9]), and we generated uniform and clustered Gaussian data. The LUCE deployment data was preprocessed to extract 4-dimensional data objects (solar panel current, global current, primary buffer voltage and secondary buffer voltage) and normalized to the values within the interval $[0, 1]$, while the synthetically generated data is also within the same interval.

The default scenario used in all experiments is the following: First we generated a set of continuous queries, either by taking a random sample from the LUCE deployment data, or by generating queries using one of the listed distributions. Second we simulated the arrival of data objects into the processing engine, either by randomly choosing data objects from the LUCE deployment data, or by generating data objects using the same distribution as for the previously generated queries. Finally, after $N$ incoming objects, we

[9] http://sensorscope.epfl.ch/

Table 3.3: Default values of parameters used in the top-k/w processing simulation.

| Parameter | Symbol | Value |
|---|---|---|
| Number of data stream objects | $N$ | $10^6$ |
| Number of active queries | $m$ | 400 |
| Number of top objects | k | 9 |
| Size of count-based sliding window | $n$ | $4 \cdot 10^4$ |
| Size of recent buffer | $b$ | 2000 |
| Data dimensionality | $d$ | 4 |
| Grid resolution | $\rho$ | 10 |
| RA: pruning coefficient | $\gamma$ | 0.2 |
| PA: probability of error | $\sigma$ | $10^{-3}$ |

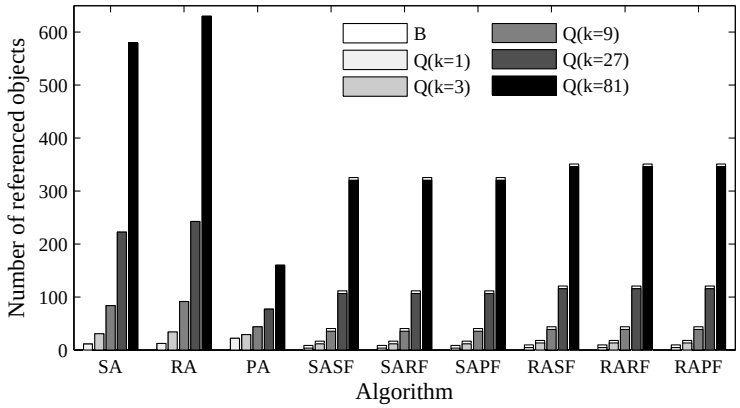
analyzed the obtained query result and processor performance. The default simulation parameters used in experiments are specified in Table 3.3.

In the following we compare space consumption and processing performance for all algorithms and examine the observed error rate of PA.

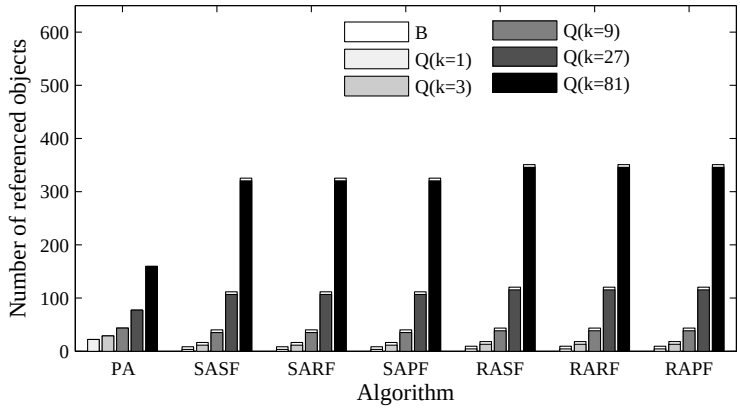
### 3.8.1 Space Consumption

In this simulation scenario we analyze space consumption of the listed algorithms without query indexing using the uniform dataset. Results with query indexing and for other two datasets are almost identical and are thus omitted. Figure 3.12 shows our results expressed as the average number of data objects referenced per query for different values of parameter $k$, where the number of such objects is denoted by $Q$. Please recall that all algorithms applying query filters use a common recent buffer for all queries which increases space consumption, and thus in our analysis we add the average number of objects stored in the buffer per query. The number of such objects equals $\frac{b}{m} = 5$ in this simulation setup and is denoted by $B$ in Figure 3.12. However, please note that in the case of a small number of queries, the number of objects stored in the buffer per query would be quite large andthus will significantly influence the space consumption.

The results in Figure 3.12a are obtained without query indexing. They clearly show that deterministic algorithms SA and RA have small space consumption only when $k$ is rather small, however, the number of objects stored in memory increases significantly for large values of parameter $k$. Compared to deterministic algorithms, PA has a substantially smaller memory footprint (4 times smaller for $k = 81$), while RA, in comparison to SA, references $\sim 10\%$ more data objects when the pruning coefficient $\gamma = 0.2$. This is in line with the complexity analysis since $\frac{\gamma}{2} \cdot (n' - k) \approx 0.1 \cdot n'$. Additionally, SA and RA reference more objects compared to their extended versions which apply query filters, while there is no significant difference on space consumption when the type of the applied filter changes. As expected, RA with query filters references on average more objects than SA with query filters. Overall PA references less objects than all deterministic algorithms for a large $k$, in particular this is true when $k > 1$ for SA and RA, and when $k > 27$ for SASF, SARF, SAPF, RASF, RARF and RAPF. For all algorithms, the space




(a) Without query indexing




(b) With query indexing

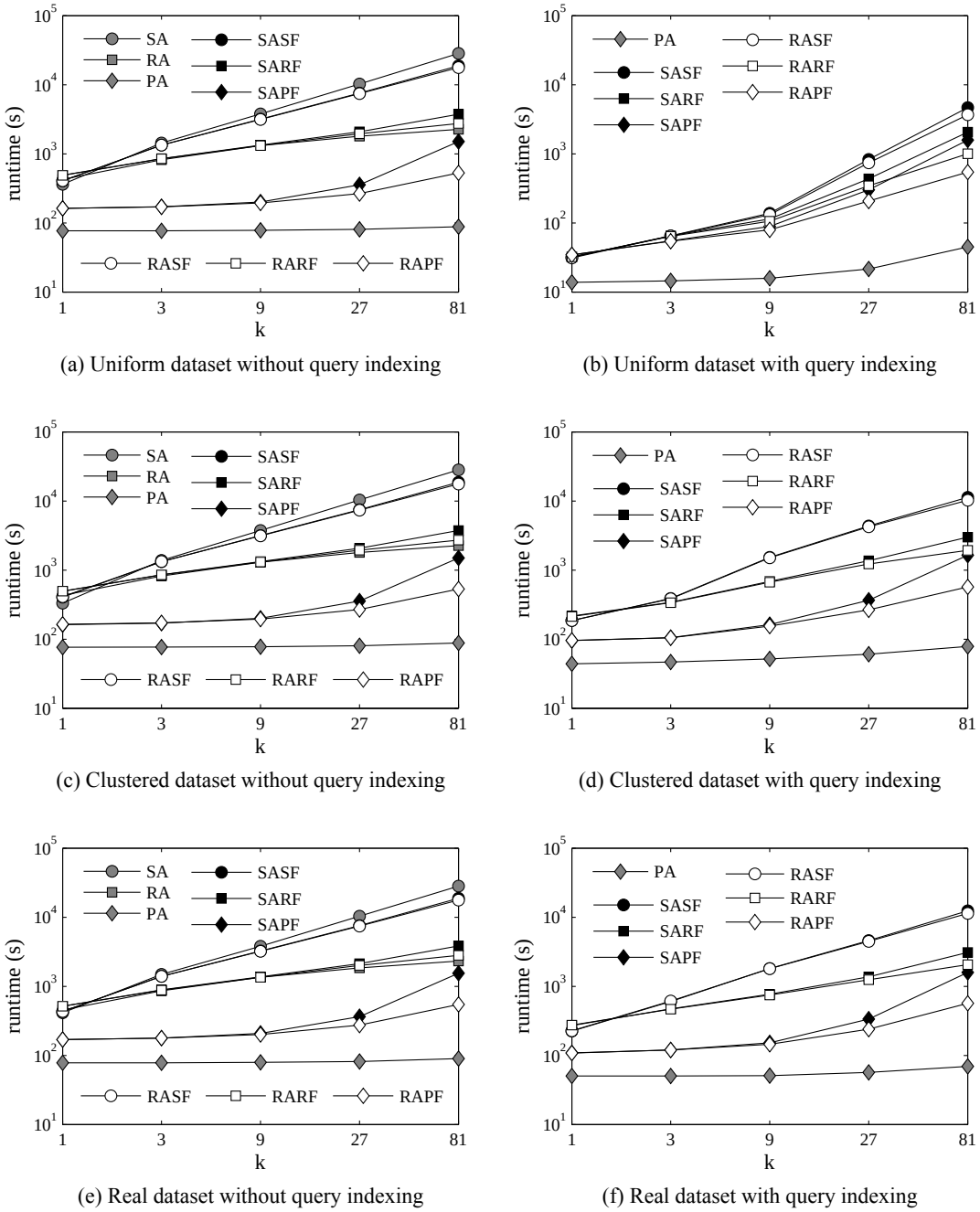
Figure 3.12: Average number of referenced data objects per top-k/w query for different query indexing methods.

complexity grows linearly with parameter $k$ as expected from the space complexity analysis presented in Section 3.7.1.

Figure 3.12b provides results with query indexing using a regular grid. We see that the query indexing only affects the number of objects referenced by the query filters, while the number of objects referenced by their queries (i.e. their query k-skybands) is not affected by indexing. This was expected from the space complexity analysis in Section 3.7.1 since the query indexing restrains the processing of low-scored data objects by the query filters, but does not affect on the number of data objects processed by the associated queries.

### 3.8.2 Processing Cost

Figure 3.13 compares the processing cost of different algorithms expressed as simulation runtime for uniform, clustered and real datasets, both with and without query indexing. We vary the parameter $k$ in all experiments, while all other parameters are set to default values as listed in Table 3.3. Figures 3.13a, 3.13c and 3.13e show the processing costs without query indexing, while Figures 3.13b, 3.13d and 3.13f




(a) Uniform dataset without query indexing

(b) Uniform dataset with query indexing

(c) Clustered dataset without query indexing

(d) Clustered dataset with query indexing

(e) Real dataset without query indexing

(f) Real dataset with query indexing

Figure 3.13: Processing cost for different datasets and query indexing methods.

show the processing cost with query indexing.

Figures 3.13a, 3.13c and 3.13e show that there is almost no difference in algorithm performance when various datasets are processed without query indexing. Furthermore, algorithms may be grouped according to processing performance as follows: SA, SASF, and RASF form the the first group of algorithms and offer the worst processing performance, especially for a large $k$. The second group of algorithms

consists of RA, SARF, and RARF, and provides better processing performance than the first group of algorithms, however, the third group of algorithms (PA, SAPF, and RAPF) outperforms the first two groups. Clearly, the processing performance of algorithms with filters is largely influenced by the type of applied filter as both SASF and RASF perform similarly to SA, while SARF and RARF perform similarly to RA. However, this is not the case for the third group of algorithms since PA, being the only probabilistic algorithm, outperforms all other (deterministic) algorithms. More importantly, the performance of PA hardly changes when $k$ increases. Please note that, for large values of the parameter $k$, the best performing deterministic algorithm is RAPF which uses RA with the probabilistic filter. Additionally, this algorithm also scales equally or better than the competing deterministic algorithms.

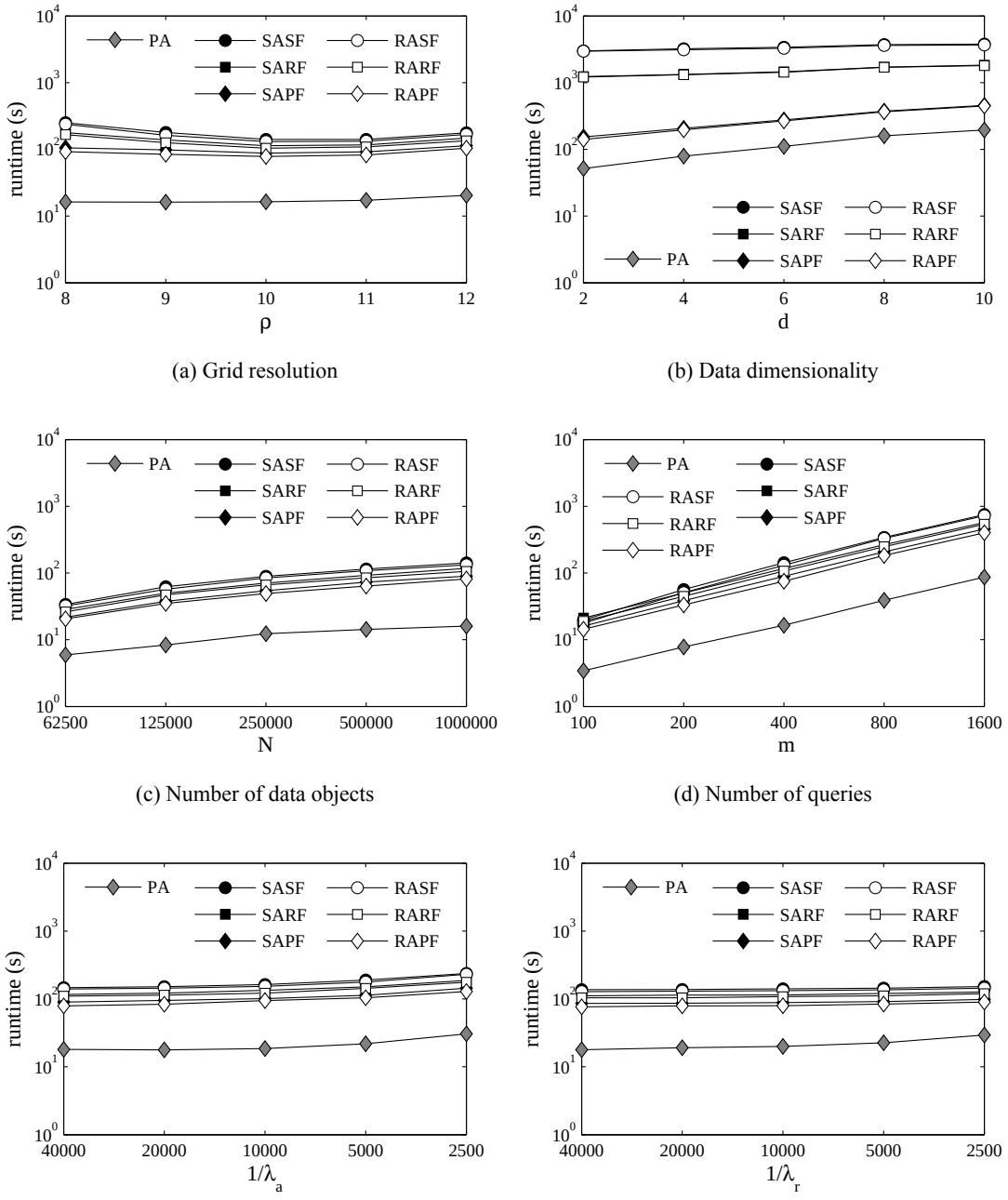
Figures 3.13b, 3.13d and 3.13f show runtime performance of different algorithms when query indexing using a regular grid is applied. As expected, the regular grid offers the best performance for the uniform dataset, while for other datasets the processing performance is only slightly improved compared to algorithms without query indexing. The explanation for this behavior is that the regular grid (which divides a Euclidean space to equally-sized cells) is adjusted for uniformly distributed data. To achieve a better processing performance in the case of non-uniform data distributions, the grid should partition the Euclidean space according to the actual data distribution. For the uniform dataset, query indexing is profitable in all situations while for the clustered and real datasets, it is only profitable if the parameter $k$ is relatively small.

In the series of experiments shown in Figure 3.14 we examine the influence of various simulation parameters on algorithm processing performance for the uniform dataset while indexing queries using a regular grid. In each of these experiments we vary one of the following nine parameters: grid resolution $\rho$, data dimensionality $d$, number of incoming data objects $N$, number of queries $m$, intensity of query activation $\lambda_a$, intensity of query replacement $\lambda_r$, recent buffer size $b$, RA pruning coefficient $\gamma$ and query window size $n$.

The results shown in Figure 3.14a analyze algorithm performance when different grid resolution $\rho$ is used for query indexing using a regular grid. These results reveal that the runtime of all algorithms is minimal for $\rho = 10$, and thus we have selected this resolution for our default experiential setup. Please note that, for a selected value of $\rho$, the grid partitions 4-dimensional Euclidean space to $\rho^4$ equally-sized cells.

The next experiment examines how data dimensionality $d$ influences algorithm performance. Figure 3.14b shows our findings when $\rho = 2$, while data dimensionality varies from 2 to 10. The runtime of PA, SAPF and RAPF increase sub-linearly when data dimensionality increases, while it scales linearly for other algorithms. It is evident that PA and deterministic algorithms with PF surpass other algorithms when processing high-dimensional data.

Figure 3.14c shows that all algorithms scale sub-linearly with an increasing number of incoming data objects $N$. This finding can be explained by the following rationale: Initially, the thresholds used for query indexing in the grid are not set and most of the incoming data objects are processed by all the queries. After the initial period, the thresholds needed for query indexing are set, and therefore the processing cost per each incoming object decreases.




(a) Grid resolution

(b) Data dimensionality

(c) Number of data objects

(d) Number of queries

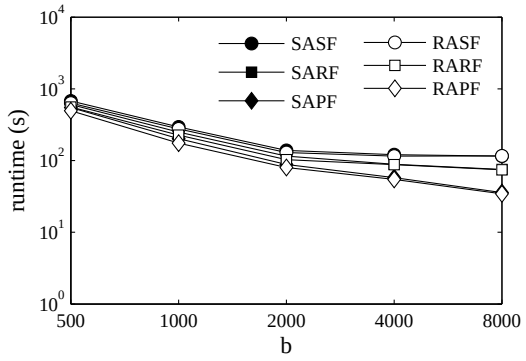
(e) Intensity of query activation

(f) Intensity of query replacement

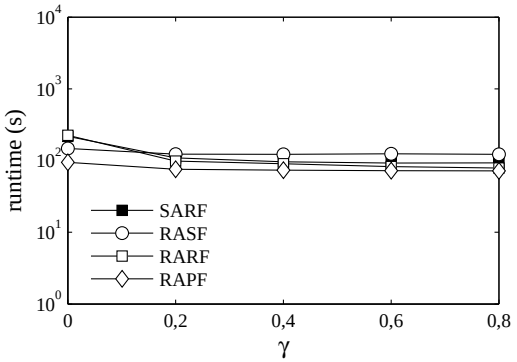
Figure 3.14: [First part] Processing cost for different values of various parameters.

Figure 3.14d examines the runtime performance of all algorithms when increasing the number of queries $m$ in the system and shows that all algorithms scale linearly with increasing number of static queries. Since grid resolution is quite high, ($\rho = 10$), the grid structure is capable to handle the increasing number of queries.

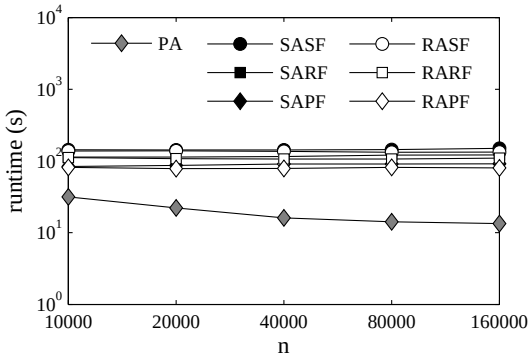
The following two experiments examine processor performance in a dynamic scenario when new




(g) Buffer size




(h) Pruning coefficient



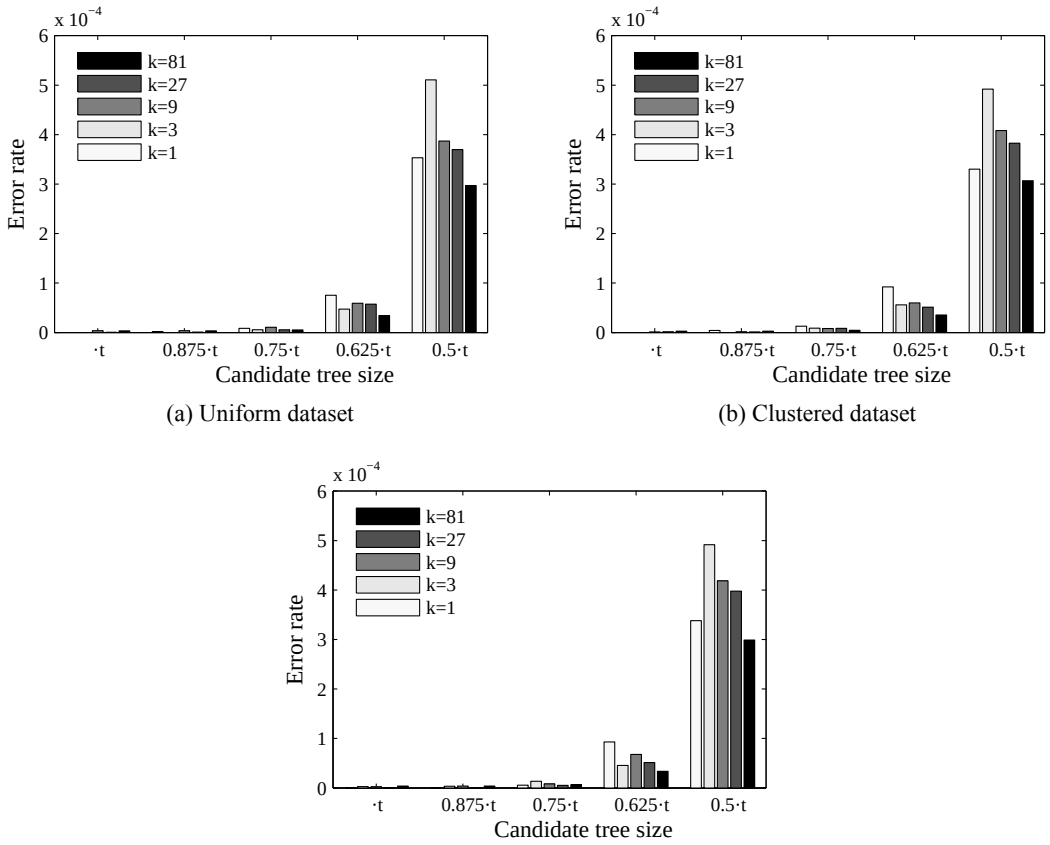

(i) Query window size

Figure 3.14: [Second part] Processing cost for different values of various parameters.

queries are added and canceled during data object processing. Figure 3.14e shows algorithm runtime when a new query is activated in the system after $1/\lambda_a$ object arrivals, where $\lambda_a$ represents the intensity of query activation. Similarly, Figure 3.14f shows experimental results when an active query is replaced by another query after $1/\lambda_r$ incoming objects, where $\lambda_r$ denotes the intensity of query replacement. We observe that all algorithms exhibit similar performance for increasing values of parameters $\lambda_a$ and $\lambda_r$ except for PA which exhibits slightly increasing runtime performance when the intensity of query replacement increases. We conclude that the regular grid is capable to efficiently handle query updates for the default setup.

The next experiment examines the influence of buffer size on the processing performance of algorithms applying buffers. Figure 3.14g shows that a larger buffer size $b$ results in more efficient processing, but this comes at the cost of an increased memory consumption. We observe that the gain in processing performance is smaller for $b > 2000$, and therefore we have selected $b = 2000$ as the default value for buffer size in our experiments.

Figure 3.14h shows the processing performance of various types of RA for different values of pruning coefficient $\gamma$. Smaller values of parameter $\gamma$ imply that pruning of dominated objects occurs more often, while larger values of $\gamma$ imply rare pruning of dominated objects at the cost of increased memory




(a) Uniform dataset

(b) Clustered dataset

(c) Real dataset

Figure 3.15: Error rate of PA for different datasets.

consumption. For $\gamma = 0$, pruning becomes strict, as in the case of SA. We have selected $\gamma = 0.2$ as the default value in our experiments since, as we can see in the figure, the gain in processing is smaller for larger values of $\gamma$.

Figure 3.14i examines algorithms performance for different query window sizes. We can see that the performance of PA improves with larger values of $n$, which is opposite to the performance of all other algorithms. The reason for such unusual finding is the increased rate of object processing for PA in case of smaller query window size because a smaller $n$ causes earlier droppings of referenced objects and thus often processing of newly arriving objects. As expected from the time complexity analysis Section 3.7.2, for all other algorithms the runtime increases when query window size increases since larger $n$ imply larger k-skyband size and therefore less efficient processing.

### 3.8.3 Error Rate of PA

PA as a probabilistic top-k/w processing algorithm produces approximate results to top-k/w queries and may erroneously report objects as potential top-k/w objects (false positives) and miss some objects that would become top-k/w objects (false negatives). Figure 3.15 shows the observed average error rate of

PA per subscription expressed as the sum of reported false positives and negatives divided by the correct number of top-k/w objects reported by a deterministic algorithm. We have run 60 simulation iterations using the default setup while varying the parameter $k$ and candidate tree size. We varied the candidate tree size from $0.5 \cdot t$ to $t$, where $t$ is the full candidate tree size that was calculated using Theorem 3.8. As we see in Figure 3.15, there is no significant difference of the observed error rate for different datasets, and almost negligible error rate is observed when the tree size varies from $t$ to $0.75 \cdot t$. Even when the tree size is $0.5 \cdot t$, the observed error rate is still much lower than the predefined theoretical upper bound $\sigma = 10^{-3}$. However, please note that all data streams used in this evaluation follow the random-order data stream model since data streams were generated by randomly selecting data objects from a selected dataset.

## 3.9 Related Work

Sliding-window query processing on data streams has received a lot of attention in the last years [52, 97, 4, 91]. We can classify existing approaches in two categories: deterministic approaches which produce correct results to defined queries, and probabilistic approaches which produce approximate results, but are in general more efficient and require less memory than deterministic approaches. In this section we review related work in the field of top-k query processing over sliding windows, and compare existing solutions to our approach. Furthermore, we identify a set of related top-k/w problems with specific types of data streams and queries.

### 3.9.1 Deterministic Top-k/w Processing

The first paper addressing the problem of deterministic top-k/w processing is [141]. The paper presents two algorithms named *skyband monitoring algorithm* (SMA) and *top-k monitoring algorithm* (TMA) for indexing top-k/w queries with aggregation scoring functions. As opposed to our approach, these authors assume that all queries are defined with the same size of the sliding window, while all data objects within the current window are stored in memory. For each query, TMA continuously maintains a self-balancing binary tree of top-k data objects from the current window, and initiates the computation of such tree from scratch in situations when the tree contains less than $k$ objects. SMA continuously maintains a k-skyband of data objects within the current window for each query represented as a linked-list in memory, and starts the computation of a k-skyband from scratch when the k-skyband contains less than $k$ objects. The authors use a two-dimensional k-skyband that does not depend on the attribute space dimensionality of data objects. After each k-skyband construction, the score of the object with rank $k$ is used as the indexing threshold in a regular grid. The algorithms process only incoming data objects within query cells of interest between two consecutive k-skyband constructions without changing the threshold. Unfortunately, the authors have not recognized that such a threshold can produce false positive and negative top-k objects. For example, if object $o_2$ in Figures 3.11a and 3.11b is dropped from the query window at a point in time when the object $o_3$ appears, the object $o_3$ will become the new top-1 object because it is inside query cells of interest, although object $o_1$ has a better score. To prevent

these types of errors, we continuously maintain the query filter indexing threshold and try to insert each data object twice into the query data structure. In their subsequent paper [143], the authors present CPM and SNN which are similar to TMA and SMA, and are developed for the problem of continuous k-NN monitoring over sliding windows.

Another work relevant to ours is [32], which proposes an algorithm for continuous k-NN monitoring on data streams with limited memory. Although the paper does not consider the sliding window queries, the monotonic expiry of data objects is similar to sliding windows. Despite the fact that the experimental evaluation shows performance of the algorithm for $k > 1$, only the implementation for $k = 1$ is presented in the paper. The presented algorithm is based on the continuous maintenance of a skyline of data objects in a linked-list. This paper is the first that introduces a recent buffer to avoid inserting of less relevant data objects to the query data structure. The query filter, called the approximate skyline, is based on the presented algorithm and uses the score of a k-th ranked object in a filter as a threshold for indexing in a regular grid. That is why the presented algorithm and filter are similar to our SA and SF, respectively.

Our SA does not apply the indexing of data objects referenced by a query, but instead keeps a linked-list of data objects in the query k-skyband. As an alternative, it is possible to use an R-tree or quadtree to index data objects in the two-dimensional (time-score) Cartesian space. For example, the BBS algorithm presented in [160] supports efficient answering to an R-tree window query that returns all points within the query window. Such a query window is a subspace of the used Cartesian or attribute space and should not be confused with the sliding window we discuss in this thesis. Upon a new object appearance, we could use the BBS to perform two window queries on R-tree, where the first query will return all objects that dominate the appeared object, while the second query will return all objects that are dominated by the appeared object. These windows (i.e. subspaces) are shown in Figure 3.3. Actually, we have evaluated this approach on the both R-trees and quadtrees, but the achieved processing performances were a few orders of magnitude worse compared to our algorithms presented in this chapter, mainly because the trees are not created for high rate of insertions and deletions. However, the thesis [204] presents an algorithm for processing of top-k queries that uses a modified R-tree to index data objects within a query window. The author is interested in the top-k queries that use aggregation scoring functions. Data object indexing is done in the hybrid multidimensional time-attributes space, and not in the time-score space. The presented experimental evaluation and complexity analysis reveals that this approach is faster for $k = 1$ than the SMA from [141], but its time complexity grows linearly with k.

The paper [71] focuses on efficient processing of ad-hoc top-k/w queries over data streams. An ad-hoc query is also interested in data objects that have appeared before the point in time of its activation. In order to perform efficient processing, the paper introduces a geometrical representation of data objects that allows using of arrangements [104]. Compared to our approach, the paper addresses a quite specific problem where the queries support only the linear additive aggregation scoring functions.

### 3.9.2 Probabilistic Top-k/w Processing

The existing probabilistic approaches typically apply either data reduction (e.g. sampling) or summaries (e.g. histograms, wavelets and synopsis) [91]. The paper [117] presents a technique for processing *e*-

approximate k-NN queries over data streams. This technique partitions the attribute space using a regular grid such that the maximum distance between any pair of points in a cell is at most $e$, and keep at most $K \geq k$ points per each cell. The indexing of points is done using B-tree and space-filling curves. The authors explain how to achieve the best accuracy with a fixed amount of memory, and the minimum memory consumption with a fixed error bound.

Our PA is based on the random-order data stream model for which any permutation of streaming objects is equally likely to appear in a data stream. This assumption is also reasonable for the secretary problem, see [64, 35, 36]. The random-order model was introduced in [150], which is one of the first papers in the field of data stream processing with limited memory. Recently, this model has attracted a lot of attention as it often describes real-world application much better than the worst-case model [101, 50, 51, 114].

### 3.9.3 Other Top-k/w Processing Problems

In this thesis we assume that the existence of incoming objects is indisputable. A different top-k/w problem, where a probability of existence is assigned to each of the incoming data objects, is analyzed in [114]. In this situation, an additional challenge comes from the exponential blowup in the number of possible worlds induced by this uncertain data model. The authors propose a series of different synopses based on data compression, buffering and exponential histograms to improve the time and space complexities of their approach.

In some application scenarios it is possible that different attribute values of the same data object are reported in separate non-synchronized data streams [103]. This model does not allow the exact score calculation for data objects. The probabilistic algorithm is based on correlation statistics of pairs of stream, which prunes more data objects than the deterministic algorithm, while keeping the necessary accuracy.

The authors of [152] present an algorithm for efficient processing of moving k-NN queries. The paper presents an incremental technique based on safe-regions called the V$^*$ diagram. This paper also surveys different approaches to the problem of static top-k/w queries over moving data objects.

## 3.10 Conclusion

In this chapter, we study the problem of processing continuous queries that monitor top-k data objects over sliding windows. This problem is not trivial because data objects which are not top-k objects within a query window at the moment of their arrival, can later on become top-k objects. Therefore a set of potential top-k objects within the query window has to be stored in memory. The existing approaches to top-k/w processing exhibit one of the following drawbacks: they support a single type of scoring functions, assume unbounded memory, or scale poorly with increasing $k$ and $n$.

In this chapter we introduce two algorithms named RA and PA. RA is a novel deterministic algorithm based on the periodical pruning of dominated data objects instead of continuous k-skyband maintenance as used by SA. PA is a probabilistic algorithm which defines a probabilistic criterion to identify potential

future top-k data objects at the time when they enter the processing system and is based on our previous work on random-order streams. The presented version of PA differs from its original version presented in [172], since we introduce a new probabilistic criterion in Theorem 3.8 which gives a better approximation of the true criterion. We believe that random-order streams are a realistic model in many data stream applications where incoming data is published by various independent sources. However, this model cannot be applied for processing correlated data coming from e.g. a single data source, or in cases when exact processing results are required. In such situations deterministic algorithms are the only option and thus we have designed a specific probabilistic filter to further improve the SA and RA implementations. The probabilistic filter implementation is based on PA, and prevents the addition of recent objects with low scores into the k-skyband when entering the processor, which improves processing performance.

The complexity analysis and experimental evaluation reveal interesting results when comparing the presented algorithms. When comparing SA and RA, the memory consumption of RA is higher than SA up to the predefined limit, however the increase of time complexity is logarithmic with $k$ for RA, while it is linear for SA. Thus, the runtime of RA smaller than the competing SA when $k > 1$, at the cost of a slightly increased, but controllable memory consumption. In all experiments, PA significantly outperforms SA and RA, both in terms of memory consumption and runtime, while it also scales better with increasing $k$ and $n$. Moreover, the observed error rate of PA is significantly smaller that the theoretical upper bound, which is another strong argument for applying PA in resource-constrained environments. In applications when errors cannot be tolerated or when data stream is not a random-order stream, the best performing algorithm is our original RAPF which requires more memory than PA (and also slightly more than SAPF), but is significantly faster than RA. Another interesting observation can be made regarding query indexing: In our particular use case scenario, the regular grid as an indexing structure for k-NN queries offer surprisingly small runtime performance improvement when clustered and real data sets are processed for larger values of parameter $k$.

# CHAPTER 4

# Subscriptions in Publish/Subscribe Systems

In Chapter 3 we formally specified two publish/subscribe systems: Boolean system and top-k/w system. According to these systems, we also distinguish Boolean subscriptions and top-k/w subscriptions, respectively. In this chapter we define and analyze these two types of subscriptions, and also discuss some very important conceptual differences between them. As we will see in the following chapters, these differences have significant implications on implementations of both centralized and distributed publish/subscribe systems. Additionally, in many publish/subscribe systems discovering and exploiting relations among individual subscriptions is an integral part of the subscription processing. Two most commonly used techniques which exploit such relations for Boolean subscriptions are subscription covering and merging. In this chapter we also show how these two techniques can be adapted to top-k/w subscriptions. Finally, we also experimentally compare Boolean and top-k/w subscriptions on the example of NN subscriptions (i.e. nearest neighbor queries) using one real and two synthetic datasets.

The rest of this chapter is organized as follows. In Section 4.1 we define and analyze Boolean and top-k/w subscriptions. We explain how these two types of subscriptions can be replicated at different network nodes in Section 4.2. In Section 4.3 we present two most commonly used techniques which exploit relations between individual subscriptions. In Section 4.4 we experimentally compare Boolean and top-k/w subscriptions. Finally, in Section 4.5 we survey different types of subscriptions which are used in state-of-the-art publish/subscribe systems.

## 4.1 Types of Subscriptions in Publish/Subscribe Systems

We classify subscriptions in publish/subscribe systems to the following three types: Boolean subscriptions, top-k/w subscriptions and other subscriptions. Boolean subscriptions are static publication filters which consider all matching publications equally relevant to a subscription, while top-k/w subscriptions are dynamic publications filters which rank publications within the subscription sliding-window. In this

Section we define and analyze these two types of subscriptions and also give a short summary of other types of subscriptions that are used in publish/subscribe systems.

### 4.1.1 Boolean Subscriptions

In this section we analyze subscriptions in Boolean publish/subscribe systems to which we refer as Boolean subscriptions. In Chapter 2 we defined the Boolean matching function, but we did not define Boolean subscriptions themselves. Therefore, we start this section by defining them.

**Definition 4.1** (Boolean Subscription)**.** Let $\boldsymbol{P}$ be a finite set of publications in a Boolean publish/subscribe system, let $c \in \boldsymbol{C}$ be a client in the system with an identifier $c_{ID} \in \mathbb{N}$, and let $m_s : \boldsymbol{P} \mapsto \{\top, \bot\}$ be a Boolean matching function. We define a double $s = \{m_s, c_{ID}\}$ as a Boolean subscription of client $c$.

The most important characteristic of Boolean subscriptions is that they are stateless subscriptions. The following lemma proves this claim.

**Lemma 4.1.** *Every Boolean subscription is a stateless subscription which is at a point in time completely defined by its double* $s = \{m_s, c_{ID}\}$.

*Proof.* The proof directly follows from the definition of the Boolean matching function in Chapter 2 since this function is defined as a time-independent function. □

Boolean subscriptions are a generalization of the three most popular types of subscriptions in publish/subscribe systems: topic-based, content-based and type-based subscriptions. These three types of subscriptions share the common characteristic that all matching publications are considered equally relevant to a subscription and do not support publication ranking. Hereafter we briefly explain their characteristics, while additional informations can be found in [81, 83, 82].

**Topic-based subscriptions** are used to specify the interest of a subscriber in a specific topic. Each publication is published on one of the available topics and subscriptions are also defined on a topic. Only those publications published on a topic selected by a subscription will match the subscription. The lack of expressiveness in specifying subscriber interest is the main drawback of this type of subscription. On the other hand, because of their simplicity, topic-based subscriptions can be processed much faster than other types of subscriptions. The hierarchy of topics is introduced to improve the expressiveness of topic-based subscriptions [155]. Each topic in a hierarchy can be derived as a more specialized topic of another one in the hierarchy. Additionally, the use of wildcards is also allowed to express cross-topic interests.

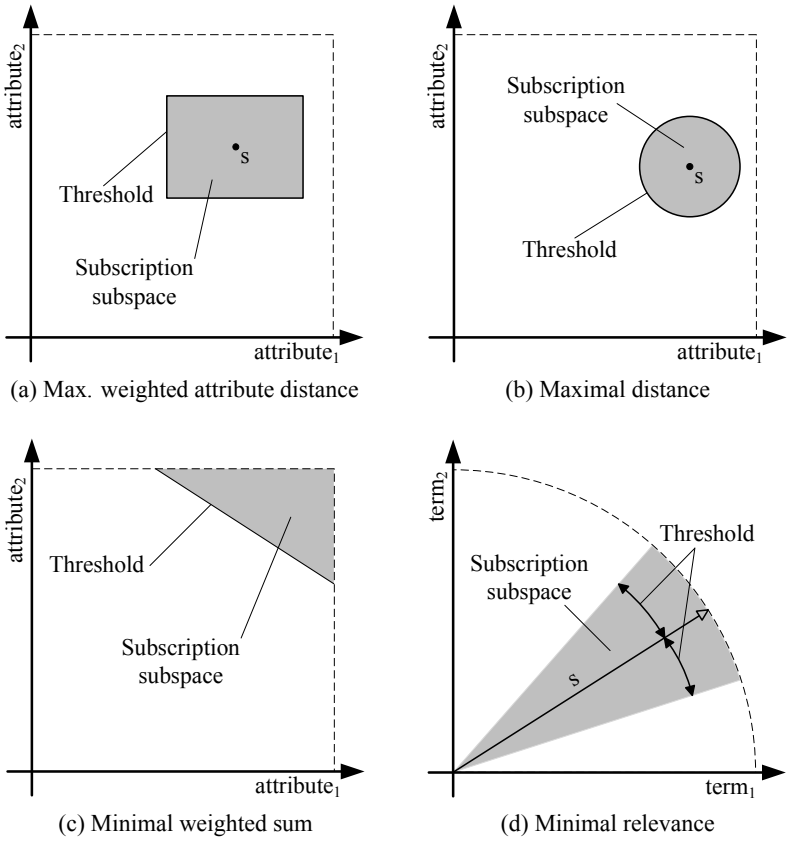
**Content-based subscriptions** describe a subscriber interest as a set of various constraints on the content of publications. These constrains can be logically combined (using and, or, etc.) and are usually defined using an application-specific subscription language. In this case, the content of each publication is defined as a set of properties (i.e. name-value pairs). A publication matches a content-based subscription when its properties satisfy constraints specified in the subscription.

**Type-based subscriptions** are somewhere between topic-based and content-based subscriptions. Each type-based subscription defines the following: 1) the type of publications in which a subscriber is interested, and 2) various constraints on the content of the public attributes of the selected type. This type of subscription is inspired by object-oriented programming languages where each object type (i.e. class) encapsulates certain attributes and methods. Analogously, a publication type can be inherited from another one. In this case, a publication matches a subscription when its type and content satisfy subscription type and constraints.

In the rest of this thesis we will focus on content-based subscriptions as the most popular type of Boolean subscriptions. We will assume that the content of publications is highly structured so that each publication is either a point in an attribute-space or vector in vector-space. Additionally, we will also assume that publication content is static, i.e. it does not change over time. Please note that these two assumptions are standard assumptions for existing content-based publish/subscribe systems, which we survey in Section 4.5. For such publications, a subscriber is usually interested in publications which belong to a subspace of the used attribute or vector-space. As such subspace is defined by the content-based subscription we will also refer to it as the subscription subspace of interest.

*Remark.* In this thesis we want to compare Boolean and top-k/w publish/subscribe systems, and thus, without loss of generality, we will assume that matching functions of Boolean subscriptions are triples consisting of a scoring function (either aggregation, distance or relevance functions), score comparator and static threshold. This assumption does not impose any restriction on the subscription language, since it only requires that each matching function, besides defining a subspace of interest, also provides a convenient way for ranking publications in this subspace. We will refer to such content-based subscriptions as *ranking Boolean subscriptions*. Therefore, a publication with a score that is better than the threshold of a ranking Boolean subscription is a matching publication for such a subscription. In other words, a threshold defines the worst publication score that will be considered as matching. This assumption allows us to compare the performance of the Boolean and top-k/w publish/subscribe systems using the same datasets, which is very important for their experimental comparison. Additionally, as stated in Chapter 3, we will assume that a scoring function is either aggregation, distance or relevance scoring function. It is very important to notice that the threshold of a ranking Boolean subscription separates the subscription subspace of interest from the rest of the available attribute or vector-space. In the next example we show four different ranking Boolean subscriptions.

**Example 4.1** (Boolean Subscriptions)**.** Figure 4.1 shows subspaces of interest for four different ranking Boolean subscriptions with the following matching functions: 1) maximal weighted attribute distance, 2) maximal distance, 3) minimal weighted sum and 4) minimal relevance. For each of these matching functions we can see a threshold that separates a subscription subspace of interest from the rest of the available space. In Figure 4.1a we see that the subspace of interest is a hyper-rectangle in the attribute space and that its sides represent the subscription threshold. It is important to mention that a subspace of interest of a content-based subscription defined in a subscription language, usually has the shape of a hyper-rectangle. Similarly, in Figure 4.1b we see that the subspace of interest is a hyper-sphere in




(a) Max. weighted attribute distance (b) Maximal distance

(c) Minimal weighted sum (d) Minimal relevance

Figure 4.1: Boolean subscriptions with different matching functions.

the attribute space, whose radius represents the subscription threshold. The subscription threshold in Figure 4.1c is a hyper-plane which divides the subspace of interest from the rest of the attribute space. Finally, the subspace of interest in Figure 4.1d is a cone-like subspace (with a convex base) of the vector space. In this case, the cosine of the half-angle at the cone top represents the subscription threshold.

### 4.1.2 Top-k/w Subscriptions

In this section we analyze subscriptions in top-k/w publish/subscribe systems to which we refer as top-k/w subscriptions. In Chapter 2 we defined the scoring function of top-k/w subscriptions, but we did not define top-k/w subscriptions themselves. Therefore, we start this section by defining the top-k/w subscription.

**Definition 4.2** (Top-k/w Subscription)**.** Let $\boldsymbol{P}$ be a finite set of publications in a top-k/w publish/subscribe system, let $u_s : \boldsymbol{P} \mapsto \{\top, \bot\}$ and $\overset{s}{\triangleright}$ be a top-k/w matching function and associated score comparator, let $c \in \boldsymbol{C}$ be a client in the system with an identifier $c_{ID} \in \mathbb{N}$, and let $k_s \in \mathbb{N}$ and $w_s \in \mathbb{R}^+$. We define a quintuple $s = \{u_s, \overset{s}{\triangleright}, k_s, w_s, c_{ID}\}$ as a top-k/w subscription of client $c$.

The most important characteristic of top-k/w subscriptions is that they are stateful subscriptions. The following lemma proves this claim.

**Lemma 4.2.** *Every top-k/w subscription s is a stateful subscription which is at a point in time completely defined by its quintuple* $s = \{u_s, \overset{s}{\triangleright}, k_s, w_s, c_{ID}\}$ *and its set of relevant publications*[1].

*Proof.* The subscription quintuple is time-independent since $k_s$, $w_s$ and $c_{ID}$ are constants, and $u_s$ and $\overset{s}{\triangleright}$ are time-independent functions by their definitions in Chapter 2. In the same chapter we defined that a finite and time-dependent set $P_s^W \subseteq \boldsymbol{P}$ of publications within the window is associated to every top-k/w subscription $s$. In Chapter 3 we proved that regardless of the used top-k/w processing algorithm, only a subset of $P_s^W \subseteq \boldsymbol{P}$ is relevant for a subscription, and since all presented algorithms reference at least this relevant set, the lemma is proven. □

The main conceptual difference between Boolean and top-k/w subscriptions is the fact that the former are stateless, while the latter are stateful subscriptions. In practice, this means that top-k/w subscriptions have to continuously keep in memory their sets of top-k and candidate publications, whereas Boolean subscriptions do not need to keep any publications in memory.

From Chapter 3 we know that every published publication is a candidate for top-k/w subscriptions employing SA and RA without filters. As a consequence, the whole attribute or vector-space is of interest for these subscriptions and thus they do not define subscription thresholds. On the contrary, top-k/w subscriptions which employ SA with filter, RA with filter or PA, define such thresholds. In the case of these subscriptions, a publication which is just published or just expelled from the recent buffer will match a subscription at a point in time if its score is better than the current threshold. In other words, such a threshold separates the subscription subspace of interest from the rest of the available space. However, while such subspaces are static in case of Boolean subscriptions, they are varying in time (i.e. dynamic) in the case of top-k/w subscriptions. The threshold of a top-k/w subscription becomes higher and contracts its subspace of interest when a newly published publication is ranked higher than k-th element of the subscription filter (for SA and RA) or its probabilistic k-skyband (for PA). Analogously, when one of k highest ranked publications in a subscription filter or probabilistic k-skyband is dropped from the subscription window, the subscription threshold becomes lower and expands its subspace of interest.

**Example 4.2** (Top-k/w Subscriptions)**.** Figure 4.2 shows subspaces of interest of four different top-k/w subscriptions with the following scoring functions: 1) weighted attribute distance, 2) distance, 3) weighted sum and 4) relevance. For each of these scoring functions we can see a dynamic threshold that separates a subscription subspace of interest from the rest of the available attribute or vector-space. As shown in the figure, if this threshold changes, the corresponding subscription subspace of interest will also change, i.e. it will either expand or contract.

### 4.1.3 Other Types of Subscriptions

Other types of subscriptions used in literature are concept-based, location-aware, XML-based, approximate, composite and parametrized subscriptions. We survey these types of subscriptions in Section 4.5.

[1] In Chapter 3 we explained that two sets (i.e. a top-k set and candidate set) are relevant for each top-k/w subscription which employs algorithms without filters, while four sets (i.e. a top-k set, candidate set, filter top-k set and filter candidate set) are relevant for top-k/w subscriptions which employ algorithms with filters.

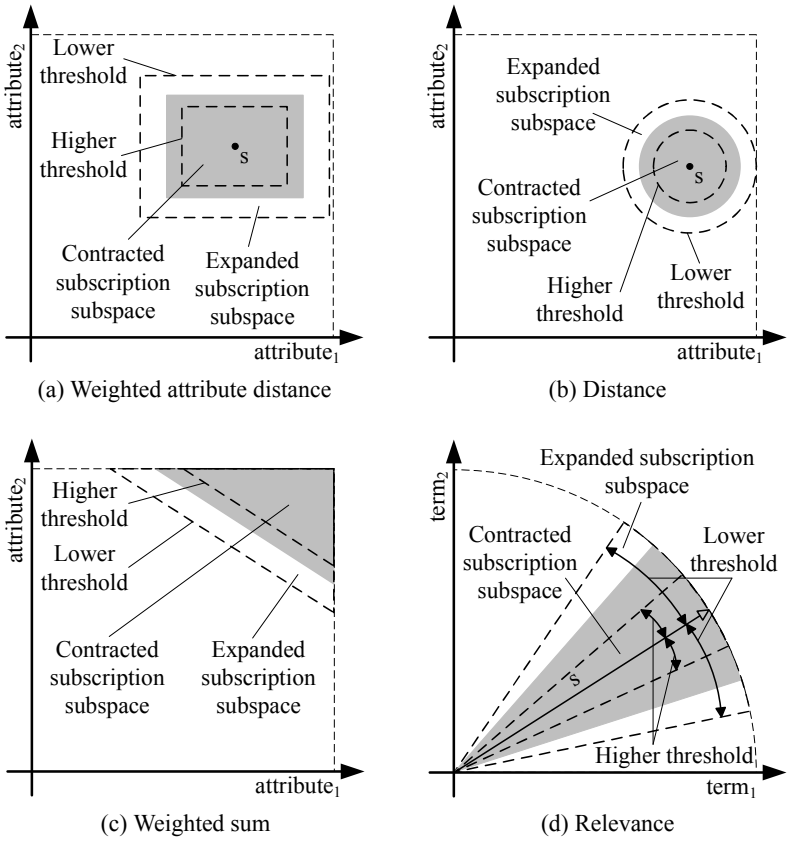



Figure 4.2: Top-k/w subscriptions with different scoring functions.

Concept-based subscriptions and corresponding publications are based on the concepts defined in an underlying ontology. Location-aware subscriptions are stateful (i.e. time-dependent) subscriptions whose matching result depends on the subscriber location. XML-based subscriptions support subscribing on semi-structured publications in the form of XML documents. Approximate subscriptions calculate degrees of match (i.e. score) between publications and subscriptions using the principles of fuzzy logic. Composite subscriptions support subscribing to temporal or causal patterns of publication occurrences which are of interest to a subscriber. Parametrized subscriptions are the most similar to our top-k/w subscriptions since they are stateful, and a state of each subscription depends on one or more dynamic parameters. We survey these types of subscriptions in Section 4.5.

## 4.2 Proxy Subscriptions

In the previous section we proved that Boolean subscription are stateless, while top-k/w subscription are stateful. This is the main conceptual difference between these two types of subscriptions. In this section we discuss some practical implications of this conceptual difference on the implementation of publish/subscribe systems.

As we know, each publish/subscribe system is distributed in its nature because it consists of a number

of clients and the publish/subscribe service. Additionally, in distributed publish/subscribe systems, the publish subscribe service is not located at one node in the network, but is distributed over a number of network nodes. Suppose that a network node activates a new subscription. Obviously, this node is aware of its own subscription and can store it in memory. However, some of the other nodes in the network should have a copy of this original subscription to decide whether a publication is of interest for this subscription. We refer to such subscription copies as *proxy subscriptions* since they represent their original subscriptions at some other network nodes different than their subscriber nodes. In the rest of this section we explain the difference between an original Boolean and top-k/w subscription and its proxy subscriptions.

### 4.2.1 Boolean Proxy Subscriptions

Every Boolean subscription is defined by its matching function and its subscriber. Since we proved in Lemma 4.1 that Boolean subscriptions are stateless, it is obvious that all proxy Boolean subscriptions contain only these two pieces of information and are thus are identical copies (i.e. replicas) of the original subscription. Therefore, once such a replica is stored on a network node, it remains consistent with its original subscription until subscription cancellation when such a replica has to be removed from the network node. In the rest of this thesis we will not make any difference between Boolean subscription and its proxy subscriptions since they are identical.

### 4.2.2 Top-k/w Proxy Subscriptions

However, a completely different situation happens in the case of top-k/w subscriptions. These subscriptions are stateful and thus their proxy subscriptions have to maintain the same state as their original subscriptions. In Lemma 4.2 we proved that the state of a top-k/w subscription is determined by its dynamic set of relevant publications. Therefore, a naive approach would be to keep the set of relevant publications consistent for a top-k/w subscription and all of its proxy subscriptions. However, this would be very impractical due to a lot of redundant information stored at nodes. Now, let us analyze what is the minimal quantity of information that we need to decide whether a publications is of interest to a top-k/w subscription. Obviously, we need to know the scoring function to calculate the publication score. Then we also need to know the current value of a subscription threshold to separate the current subspace of interest from the rest of the used attribute or vector space. Finally, we also need to know the score comparator to correctly identify the subscription subspace of interest. In other words, to decide whether a top-k/w subscription is interested in a publication, and if we know the current values of the three previously mentioned parameters, we will not need to know the values of subscription parameters $k$ and $w$, nor the elements of its set of relevant publications. In the next definition and theorem, we formally define the proxy of a top-k/w subscription and also formally prove that the listed parameters are sufficient to correctly decide whether this subscription is interested in a publication or not. Please recall that we have used the same parameters for indexing of top-k/w subscriptions in Chapter 3.

**Definition 4.3** (Top-k/w Proxy Subscription)**.** Let $s = \{u_s, \overset{s}{\triangleright}, k_s, w_s, c_{ID}\}$ be a top-k/w subscription of a

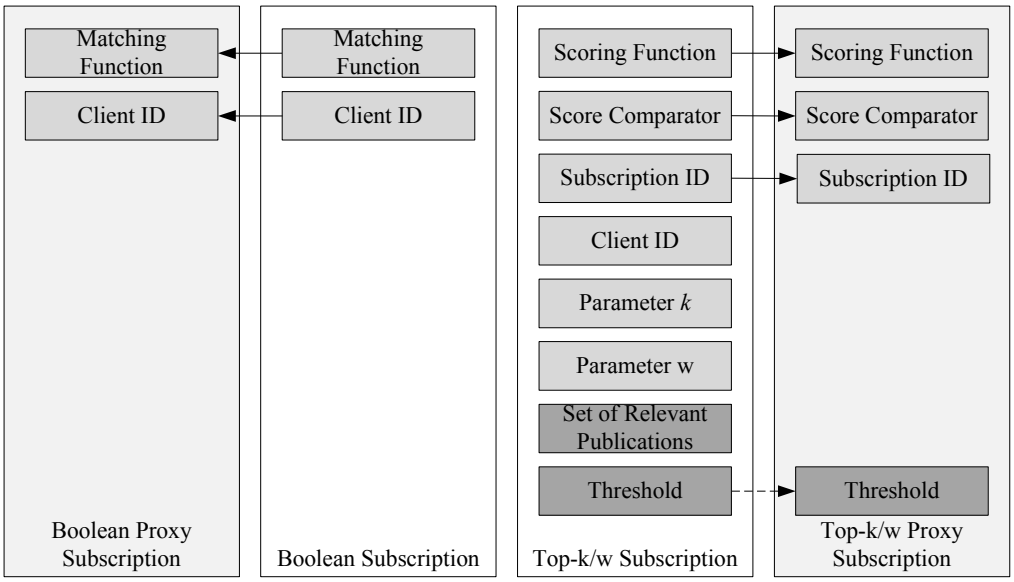



Figure 4.3: Components of the Boolean, top-k/w subscription and their proxy subscriptions.

client $c \in \boldsymbol{C}$ with an identifier $s_{ID}$ that is at a point in time $t$ in a state $\varsigma_s(t)$ in which its threshold has a value $ts_s(t)$. We define a proxy subscription of $s$ as quadruple $s^P = \{u_s, \overset{s}{\triangleright}, ts_s(t), s_{ID}\}$.

**Theorem 4.3.** *Any network node which at a point in time t has a proxy subscription* $s^P = \{u_s, \overset{s}{\triangleright}, ts_s(t), s_{ID}\}$ *synchronized with its original top-k/w subscription* $s = \{u_s, \overset{s}{\triangleright}, k_s, w_s, c_{ID}\}$*, can correctly decide whether a publication which has just been published or expelled from the recent buffer is of interest for s or whether it is not.*

*Proof.* The proof directly follows from Theorems 3.10 and 3.9 because in both cases, $p$ would not be of interest for $s$ if it is scored worse than threshold $ts_s(t)$ of $s$. □

Since scoring functions and score comparators are static (i.e. time-independent), we only need to synchronize thresholds to make top-k/w proxy subscriptions consistent with their originals. In practice, the thresholds always fluctuate in time around a central value, and this fact can be used for efficient implementations of distributed top-k/w systems as we will see in the following chapters. It is important to notice that processing[2]a publication for a top-k/w proxy subscription is less computationally intensive task than processing the same publication for the original top-k/w subscription since it only consists of score calculation and score comparison with the subscription threshold. To distinguish between these two types of processors, we refer to the former as the *lightweight top-k/w processor*, and to the latter as the *heavyweight top-k/w processor*. Additionally, the former requires less memory since it does not involve storing of candidate publications. However, in a distributed publish/subscribe system, the recent buffer is also distributed in the network, and thus some nodes also have to store buffered publications in their memory. We will analyze such systems in detail in the following chapters.

[2]Please note that exactly the same type of processing is needed for ranking Boolean subscriptions.

Figure 4.3 shows different components of the Boolean, top-k/w, and their proxy subscriptions. We see that Boolean proxy subscription is identical to the original Boolean subscription since both of them define two pieces of information: the static Boolean matching function and static subscriber identifier. We also see that the top-k/w subscription is composed of eight pieces of information: the static scoring function, static score comparator, static subscription identifier, static subscriber identifier, static parameters $k$ and $w$, dynamic threshold, and dynamic set of relevant publications. Please recall that the value of such a dynamic threshold directly depends on the set of relevant publications. Finally, we also see that the top-k/w proxy subscription is composed of the following four pieces of information: the scoring function, score comparator, original subscription identifier and dynamic threshold. The values of dynamic thresholds have to be synchronized between original top-k/w subscription and its proxy subscriptions.

## 4.3 Relations Between Subscriptions

In this section we present the two most important techniques which exploit relations between individual subscriptions. These two techniques are subscription covering and subscription merging. We first present and explain each technique for Boolean subscriptions and later adapt it to top-k/w subscriptions. We also present several examples which demonstrate subscription covering and merging for Boolean subscriptions.

### 4.3.1 Subscription Covering

A Boolean subscription is *covered* by another Boolean subscription if every publication which matches the former subscription also matches the latter subscription. Additionally, a subscription is covered by a set of subscriptions when each publication which is matching for this subscription is also matching for at least one of the other subscriptions. In the second situation it is possible that a subscription is covered even when neither one of the other subscriptions is covering it completely. Please note that the covering relation is non-commutative since it does not imply that subscription $s_1$ which is covered by subscription $s_2$ also covers $s_2$. Deciding whether a Boolean subscription is covered by a set of previously defined subscriptions was proven to be co-NP complete [193]. In a Boolean publish/subscribe system, the forwarding of a covered subscription to the network (i.e. to the system) is redundant, because its subspace of interest is already included in subspaces of subscriptions that cover it.

**Example 4.3** (Subscription Covering)**.** Figure 4.4 shows a triplet of ranking Boolean subscriptions for each of the matching functions from the previous example. In each of the four subfigures, subscription $s_1$ covers subscription $s_2$ because its subspace of interest contains (i.e. covers) the subspace of interest of $s_2$. Thereby, if subscription $s_1$ precedes subscription $s_2$ in time, subscription $s_2$ should not be forwarded to the network, because it does not carry any new information. Additionally, each of the four pairs of subscriptions $s_1$ and $s_3$ have a mutual subspace of interest, but neither subscription covers the other.

In the case of Boolean subscriptions, a previously covered subscription will become uncovered when all of the subscriptions which have been covering it are canceled by their clients. The opposite situa-

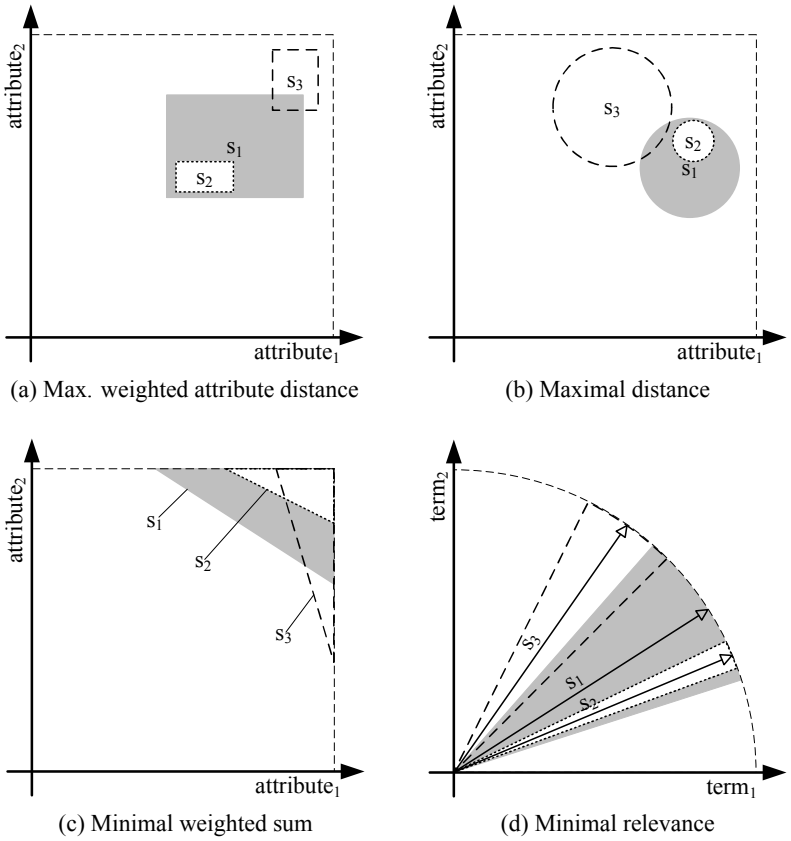



Figure 4.4: Covering of Boolean subscriptions for different matching functions.

tion is also possible as a previously non-covered subscription can become covered by a newly activated subscription. Therefore, when subscription covering is used in Boolean publish/subscribe systems, the covering relations between active subscriptions should be checked after every subscription activation and cancellation.

The covering relation between top-k/w subscriptions is also possible, but it is different from the case of Boolean subscription covering. A top-k/w subscription certainly covers another top-k/w subscription if they have identically defined scoring functions, score comparators and parameters $w$, and if the former subscription defines a larger value of the parameter $k$ than the latter subscription. However, such a covering is actually a very rare situation in real-life top-k/w systems, and thus is not very useful in practice. However, we can check instead if the subspace of interest of one top-k/w proxy subscription covers the subspace of interest of another. Analogous with Boolean subscriptions, cancellations and activations of top-k/w subscriptions may also cause changes in covering relations between active top-k/w proxy subscriptions. However, this can also happen when the subspace of interest of a top-k/w proxy subscription expands or contracts due to its threshold change. For this reason, the covering relations between active top-k/w proxy subscriptions should be checked after every top-k/w subscription threshold change, and every subscription activation and cancellation. Since the thresholds change frequently, the usability of this technique for this type of subscriptions also comes in question.

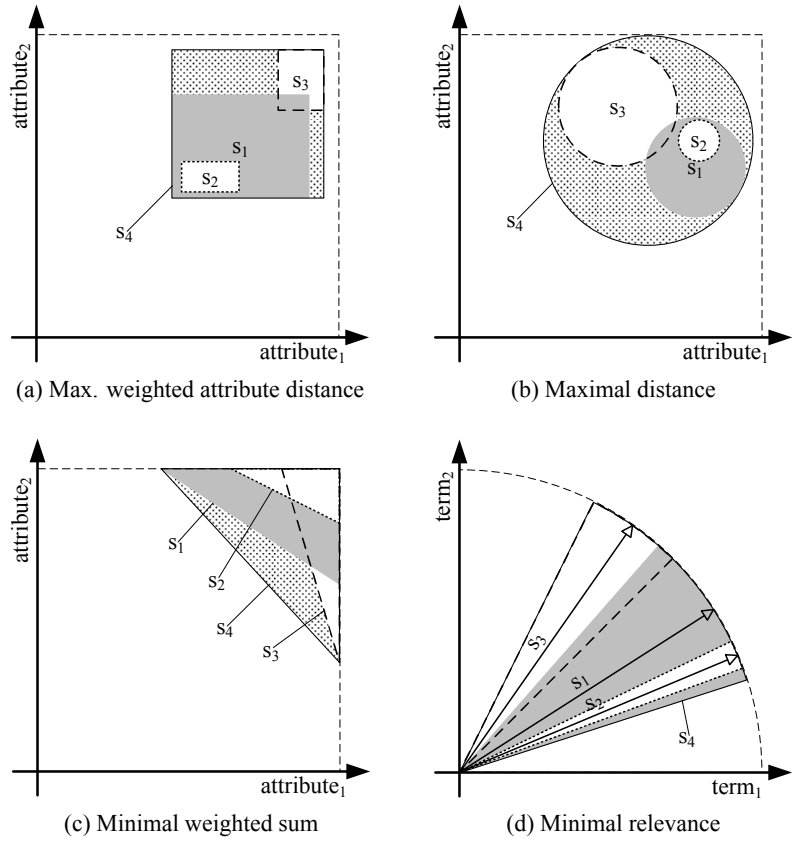



Figure 4.5: Merging of Boolean subscriptions for different matching functions.

### 4.3.2 Subscription Merging

Two or more Boolean subscriptions can be *merged* together in a broader subscription which then covers all of the original subscriptions. Only the resulting merged subscription (*merger*) can be forwarded to the network (i.e. to the system), where it replaces the original subscriptions. Subscriptions can be merged together either in a perfect or imperfect way. Any part of the subspace of interest of a perfect merger is covered by at least one of the original subscriptions, whereas this does not have to be true for imperfect mergers. Both perfect and imperfect merging reduce subscription information stored at nodes because every merger replaces two or more original subscription entries. However, imperfect merging can increase network traffic since some non-matching publications, which belong to a subspace that is not covered by original subscriptions, can be forwarded to network.

**Example 4.4** (Subscription Merging)**.** Figure 4.5 shows mergers for triplets of ranking Boolean subscriptions for each of the matching functions from the previous two examples. In each of these subfigures, subscription $s_4$ represents the merger for subscriptions $s_1$, $s_2$ and $s_3$. The triplets of subscriptions which are shown in Figure 4.5a, Figure 4.5b and Figure 4.5c have imperfect mergers since the dotted areas are not covered by any of original subscriptions. On the contrary, the triplet of subscriptions in Figure 4.5d is perfectly merged together.

The merging of subscriptions also exists in the case of top-k/w proxy subscriptions. However, after each expansion of a merged top-k/w proxy subscription, we need to check if it has become interested in a subspace which is not covered be its merger. When this is true, we have to forward such proxy subscription in the network since it is no more covered by the merger. Similarly, after each contraction of a top-k/w subscription, we should check whether it is possible to contract its merger. Analogous with the case of subscription covering, and due to the frequent changing of top-k/w subscription thresholds, the usability of this technique for this type of subscription is questionable.

## 4.4 Experimental Evaluation

In this section we present an experimental study comparing the number of delivered publications for (ranking) Boolean and top-k/w subscriptions. For the top-k/w subscription processing we use two algorithms presented in Chapter 3: PA and RAPF. The former algorithm is probabilistic, while the latter is the best performing of deterministic top-k/w processing algorithm.

Analogous to the experimental study in Chapter 3, we have selected NN subscriptions (i.e. NN queries) as a use case for our evaluation. Both subscriptions and publications in our experiments are represented as points in a $d$-dimensional Euclidean space. The score of a publication $p$ with respect to subscription $s$ is calculated using the following formula: $u_s(p) = d(p_s, p_p) = [\sum_{i=1}^{d}(v_i - v_i)^2]^{\frac{1}{2}}$, where $p_p = \{v_1, v_2, \ldots, v_d\}$ and $p_s = \{v_1, v_2, \ldots, v_d\}$ are points representing $p$ and $s$, respectively. The score comparator $\overset{s}{\rhd}$ is defined such that lower scores imply higher ranks. Additionally, Boolean subscriptions define static thresholds, whereas top-k/w subscriptions define parameters $k$ and $w$.

Please note that for the sake of simplicity, we omit subscripts of subscription parameters in this section, if otherwise not explicitly stated.

We have also used one real and two synthetic datasets in the experimental evaluation. In particular, the real dataset used in experiments is the LUCE deployment data (environmental data collected from large-scale wireless sensor networks within the project SensorScope[3]), and we generated uniform and clustered Gaussian data. The LUCE deployment data was preprocessed to extract 4-dimensional data objects (solar panel current, global current, primary buffer voltage and secondary buffer voltage) and normalized to the values within the interval $[0, 1]$, while the synthetically generated data is also within the same interval.

The default scenario used in all experiments is the following: First we generated the set of subscriptions, either by taking a random sample from the LUCE deployment data, or by generating subscriptions using one of the listed distributions. Second we simulated the publishing of publications, either by randomly choosing publications from the LUCE deployment data, or by generating publications using the same distribution as for the previously generated queries. Finally, after $N$ publications, we analyzed the obtained results. The default simulation parameters used in experiments are specified in Table 4.1.

In the following we examine the observed average thresholds for top-k/w subscriptions, average top-k scores for PA and RAPF, and compare the average number of delivered publications per subscription

[3] http://sensorscope.epfl.ch/

Table 4.1: Default values of parameters used in the subscription comparison simulation.

| Parameter | Symbol | Value |
|---|---|---|
| Number of publications | $N$ | $10^6$ |
| Number of subscriptions | $m$ | 400 |
| Size of recent buffer | $b$ | 2000 |
| Data dimensionality | $d$ | 4 |
| PA: probability of error | $\sigma$ | $10^{-3}$ |

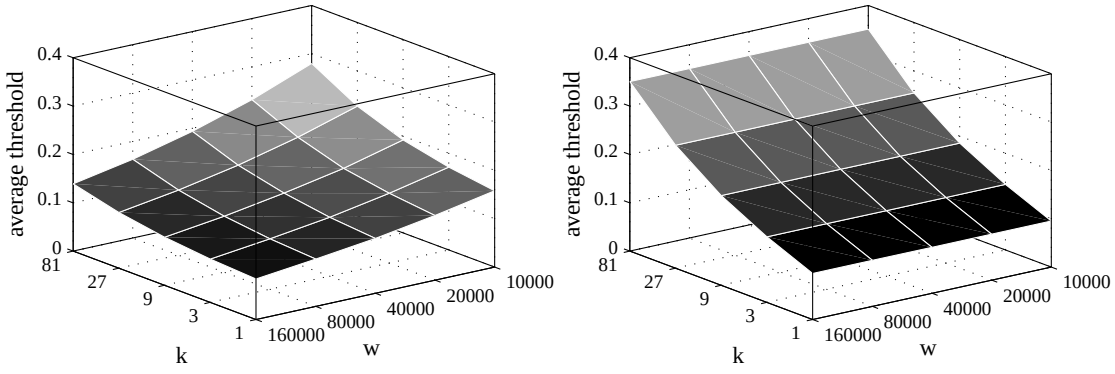
for Boolean and top-k/w subscriptions. Please recall that the average threshold for PA is the score of the candidate publication with the worst rank, while for RAPF it is the score of the k-th ranked publication in the (probabilistic) filter.

### 4.4.1 Average Threshold of Top-k/w Subscriptions

In this simulation scenario we compare average thresholds of PA and RAPF for different datasets. These thresholds are used for subscription indexing, as explained in Chapter 3. The average thresholds are shown in Figure 4.6. Please note that the scales of y-axes are different for different datasets. As we can see, average thresholds of PA are always lower than of RAPF. This is expected since PA does not use the recent buffer, while RAPF strongly depends on it. Actually, the average threshold of RAPF does not depend on the subscription window size, but on the recent buffer size which was constant in all experiments. We can also see that the average threshold increases with increasing parameter k for both algorithms, and for PA increases with decreasing parameter w. Since threshold surfaces are similar for different datasets we conclude that top-k/w subscriptions adapt well to a given dataset. Please note that the larger subscription threshold means a larger number of subscription cells of interest which then results in a less efficient processing.

### 4.4.2 Average Top-k Score of Top-k/w Subscriptions

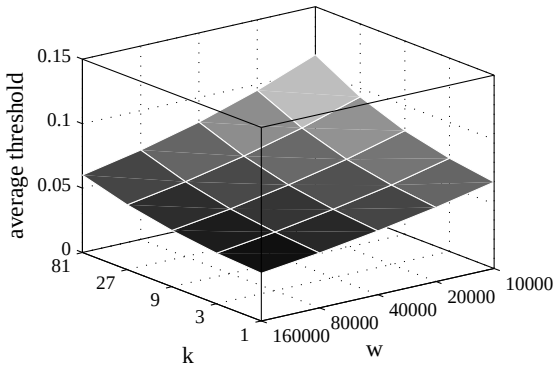
In the next experiment we analyze the average top-k score (i.e. score of the k-th ranked referenced publication) of top-k/w subscriptions for different datasets. The purpose of this experiment is to identify the values of Boolean subscription thresholds which we will later use for the comparison of Boolean and top-k/w subscriptions. If the threshold of a Boolean subscription is equal to the average top-k score of a top-k/w subscription, these two subscriptions will have similar number of matching publications. Figure 4.7 shows our results expressed as the average score of k-th ranked of referenced publications. Please note that these average scores are identical for all deterministic top-k/w processing algorithms and almost identical for PA since it has very low error rate as shown in Chapter 3. For this reason we show only the average top-k scores for PA. Again, the scales of y-axes are different for different datasets. We can see that the average top-k scores are different for different datasets, which is expected since such scores have to be lower for more clustered datasets, and that these scores increase with increasing parameter $k$ and decreasing parameter $w$. The shapes of average top-k scores are similar for different datasets which



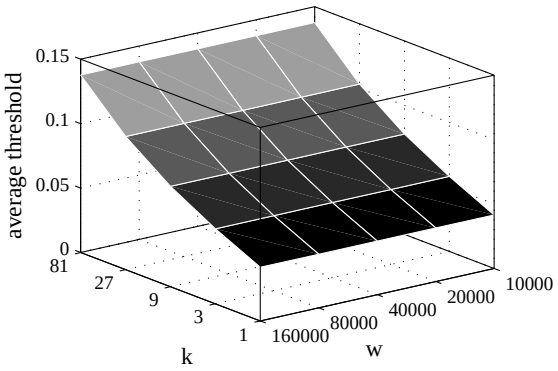

(a) PA: uniform dataset



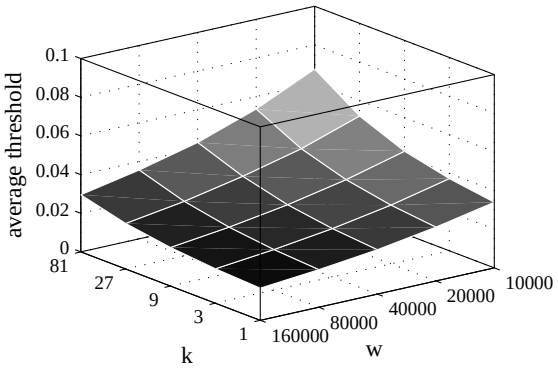

(b) RAPF: uniform dataset



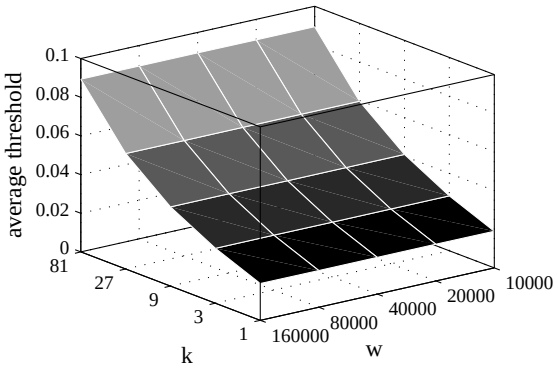

(c) PA: clustered dataset




(d) RAPF: clustered dataset




(e) PA: real dataset



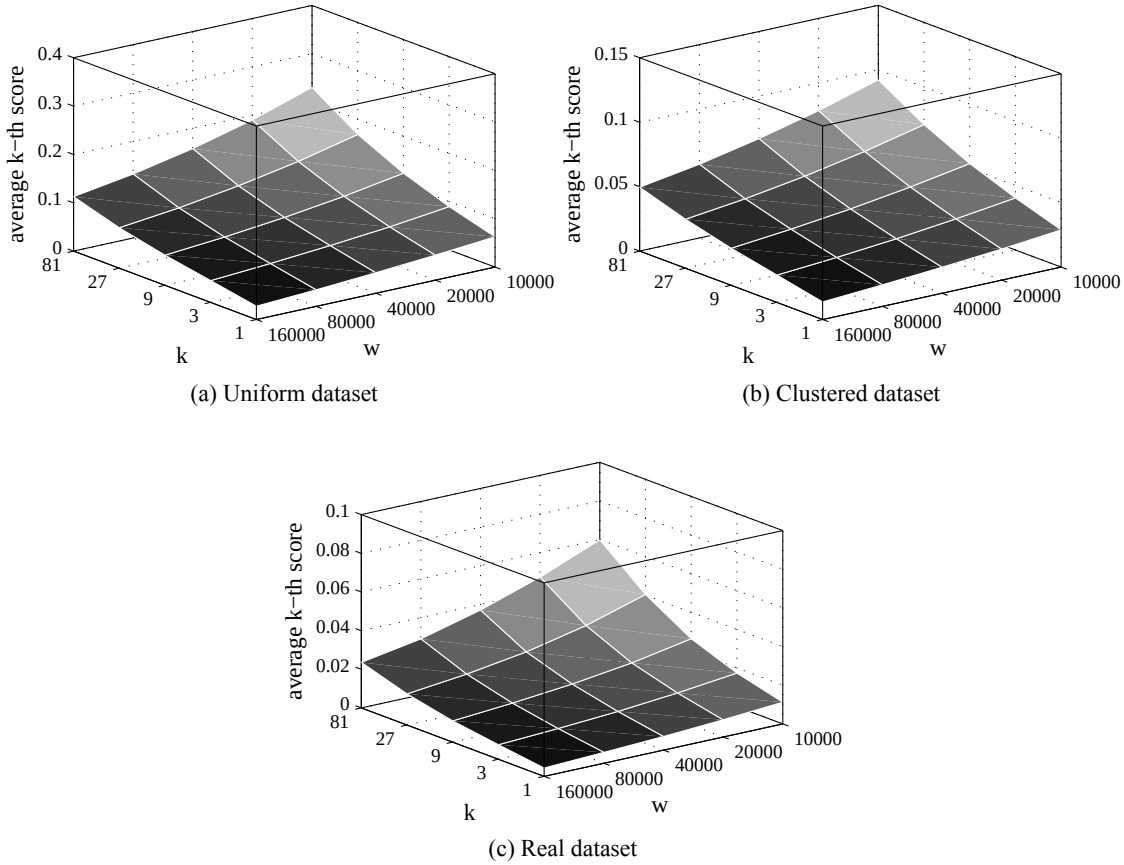

(f) RAPF: real dataset

Figure 4.6: Average top-k/w subscription threshold for different datasets.

again implies that top-k/w subscriptions very well adapt to a given dataset.

### 4.4.3 Average Number of Matching Publications per Subscription

In this experiment we analyze the number of matching publications per subscription for Boolean and top-k/w subscriptions. As we can see in Figure 4.8, due to the adaptability of top-k/w subscriptions to




(a) Uniform dataset

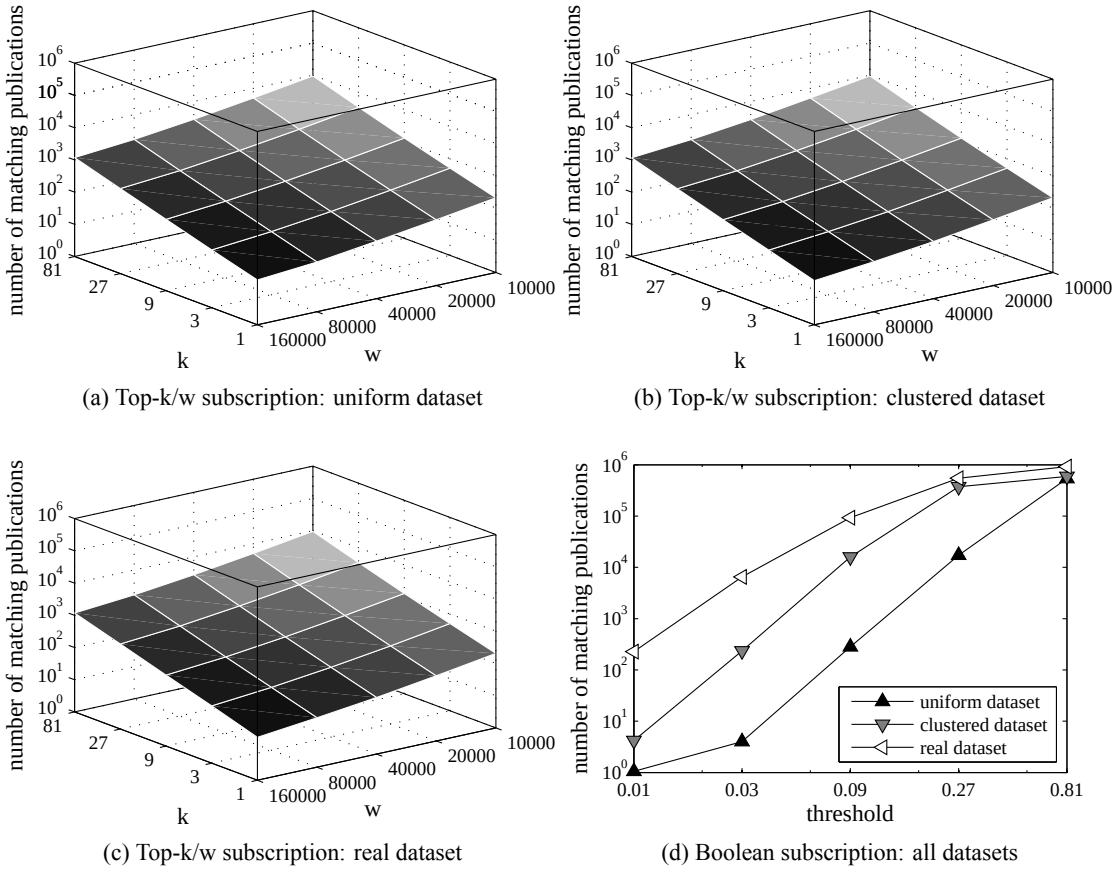
(b) Clustered dataset

(c) Real dataset

Figure 4.7: Average top-k score of a top-k/w subscription for different datasets.

different publication datasets, the average number of matching publications per top-k/w subscriptions does not depend on the dataset. This evaluation experimentally proves that the top-k/w model performs effective filtering regardless of the dataset. As expected, when increasing $w$, the number of matching publications decreases, while it increases when increasing $k$. On the contrary, the number of matching publications for Boolean subscriptions heavily depends on the dataset (and selected threshold value). As we can see, a small change in the Boolean subscription threshold results in a large variation of the number of matching publications for different datasets. This shows that Boolean subscriptions do not provide any means for controlling the number of delivered publications per subscription. Please note that the Boolean threshold value of 0.09 is analogous to the average top-k score of top-k/w subscription with parameters $k = 9$ and $w = 40000$ for the uniform dataset as shown in Figure 4.7a, and these subscriptions thus have similar numbers of matching publications for these values of parameters.

The setup of the last experiment is similar to the previous one since we analyze the average number of matching publications per subscription for Boolean and top-k/w subscriptions while varying the publication intensity $\lambda$. In this experiment we are using only the uniform dataset and the default values of all parameters specified in Table 4.1. The obtained simulation results are shown in Figure 4.9. We can see that similarly to the previous experiment, and due to the adaptability of the top-k/w model to different




(a) Top-k/w subscription: uniform dataset

(b) Top-k/w subscription: clustered dataset

(c) Top-k/w subscription: real dataset

(d) Boolean subscription: all datasets

Figure 4.8: Average number of matching publications per Boolean and top-k/w subscription for different datasets.

publication intensities, the average number of matching publications per top-k/w subscription does not depend on the parameter $\lambda$, while for Boolean subscriptions, on the contrary, this number heavily depends on the parameter $\lambda$ since it increases linearly with the increasing parameter $\lambda$.

### 4.4.4 Conclusion

Our experimental evaluation has shown that top-k/w subscriptions adapt very well both to the distribution of publications in their attribute space and the publication intensity. Additionally, it has also shown that, on the other hand, the number of matching publications for Boolean subscriptions heavily depends on the publication distribution and intensity, which are generally unknown in advance.

## 4.5 Related Work

In this section we give an overview of the types of subscriptions that are considered in both research literature and industry solutions. Since the techniques of subscription covering and merging are related

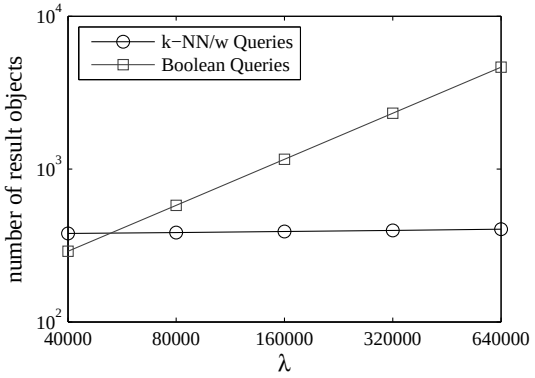



Figure 4.9: Average number of matching publications per Boolean and top-k/w subscription for different publication intensities.

to distributed publish/subscribe systems, we survey related work in this area in Chapter 6.

### 4.5.1 Boolean Subscriptions in Publish/Subscribe Systems

From the historical point of view, the first publish/subscribe systems supported only topic-based subscriptions. The examples of such systems are TIB/RV [155], Vitria [192], Corba Event Service [153, 176], iBus [10], Bayeux [209] and Scribe [182, 49]. TIB/RV [155] is the first topic-based publish/subscribe system with support for wildcards, a convenient way to subscribe to several topics with a single subscription. This system also supports a flexible topic hierarchy. Corba Event Service [153, 176] is a specification from the Object Management Group which does not support topic hierarchy. PrismTech OpenFusion Notification Service and IONA Orbix Notification are examples of systems implementing this specification. iBus [10] and Vitria [192] are simple multicast-oriented topic-based publish/subscribe systems. Bayeux [209] is an application-level multicast system, which organizes the multicast receivers into a distribution tree rooted at the source, and is built on top of Tapestry [205], a wide-area location and routing architecture. Scribe [182, 49] is a large-scale event notification infrastructure, built on top of Pastry [181], a generic peer-to-peer object location and routing scheme for a large-scale overlay network of nodes connected via the Internet. Since the scalability is much more important for Bayeux and Scribe than expressiveness, these systems do not support the topic hierarchy.

Currently, the most popular types of subscriptions in publish/subscribe systems are content-based subscriptions. The examples of publish/subscribe systems which support these subscriptions are: Yeast [120], Elvin3 [187], Elvin4 [186], GEM [137], Keryx [201], Corba Notification Service [154, 176], READY [100], Gryphoon [24, 6], SIENA [45], Le Subscribe [161], JEDI [69], Kyra [40], Meghdoot [102] and Rebeca [149]. Yeast [120] is a centralized publish/subscribe system with a rich subscription language. Besides of a number of predefined classes and attributes, which are used for specifying of publications, Yeast also supports defining of additional classes and attributes. Elvin4 [186] is an improved version of Elvin3 [187]. The improved version supports the following types of publication attributes: integer, string, floating point and binary data such as images or compiled code. The subscription lan-

guage of Elvin4 is very rich and supports both the advanced arithmetic and string matching constraints. GEM [137] is a distributed publish/subscribe system which uses a declarative and interpreted subscription language. This language is based on the assumption of a global system time. Keryx [201] is a distributed publish/subscribe system which enables support for Distributed Virtual Environments. In this system, publications are specified in Self-Describing Data Representation, which is a textual syntax for structured data presented in [130]. Corba Notification Service [154] is another specification from the Object Management Group. This specification supports a rich constraint language which allows defining of type and content-based subscriptions. For example, the subscription language of READY [100] follows this specification, but also supports grouping constructs, composite events and event aggregation. Gryphoon [24, 6] is a distributed publish/subscribe system with a simple subscription language in which each subscription is a conjunction of a set of simple constraints on one or more publication attributes. The subscription language used in SIENA [45] supports specifying of a set of attribute-constraint pairs, where each constraint is a tuple specifying: a type, a name, a binary predicate operator, and a value for an attribute. Le Subscribe [161] is a publish/subscribe system which is well suited for information published on the Web. This system uses a semi-structured publication model and supports subscriptions which are a conjunction of predicates, where each predicate is a triple of the form attribute-operator-element. The subscription language supports arithmetic operators and two specific operators: *kind of* and *contains*. Publications in JEDI [69] are simple sequences of strings, while subscription language is very simple and supports only regular expressions. Meghdoot [102] uses a data model proposed by Fabret et al. [85]. In this model, a subscription is a conjunction of predicates (i.e. simple range or value constraints) over one or more attributes. The subscription language of Rebeca [149] is very simple: a publication is a set of name-value pairs, while a subscription is a conjunction of constraints on single name-value pairs. The specification [107] of Java Message Service (JMS) defines that a publication (JMS message) consists of a header, a set of properties and a body. Subscriptions (JMS message selectors) allow clients to specify the characteristics of publications they are interested in. Only the header and properties are checked during the matching process, while publication body is ignored. JMS subscription language is rich since each subscription is a conditional expression that supports logical, comparison and arithmetic operators.

Eugster [81] introduced type-based subscriptions in publish/subscribe systems. These systems offer a much better integration of object-oriented programming languages and publish/subscribe systems. Hermes [165] is an example publish/subscribe middleware with support for type-based subscriptions.

### 4.5.2 Other Types of Subscriptions in Publish/Subscribe Systems

Alternative types of subscriptions are: concept-based [63, 171], location-aware [138, 88, 191, 68], XML-based [53, 54, 191] and approximate [128, 129, 8, 164, 42, 124] subscriptions. Approximate subscriptions are the most related to our work since such subscriptions calculate scores of publications (degrees of match) using the principles of fuzzy logic. This enables partial matching between publications and subscriptions, which eliminates situations with a small number of matching publications. However, since the authors use fixed score thresholds and do not rank publications among each other, such subscriptions may produce situations where subscribers are flooded with too many matching publications.

Additionally, some publish/subscribe systems with content-based subscriptions also support subscribing to composite events. A composite event is a (temporal or causal) pattern of publication occurrences which is of interest to a subscriber. Examples of such systems are Yeast [120], GEM [137], READY [100], SIENA [45], EPS [139], PADRES [126, 125] and Cayuga [73]. Subscriptions in such systems are also stateful (as our top-k/w subscriptions) since several publications are usually involved in the matching of a single subscription pattern, and thus partial matching states have to be stored in memory. For example, Cayuga[73] is a stateful publish/subscribe system based on nondeterministic finite state automata (NFA) whose subscription language supports composite events using several binary operators (i.e. projection, selection, renaming, union, conditional sequence, iteration, aggregate) and powerful language features such as parameterization and aggregation. Our work, however, differs significantly from these approaches since top-k/w subscriptions do not support subscribing to composite events.

Parametrized subscriptions introduced in [111] are very similar to our top-k/w subscriptions since they also have a static and dynamic part. In case of such subscriptions, the dynamic part consists of one or more parameters and the matching between publications and a parametrized subscription depends on the subscription parameters. Therefore we could say that our top-k/w proxy subscriptions is a parametrized subscription with a dynamic threshold as its only parameter.

In Chapter 3 we surveyed the relevant work in the field of top-k/w processing over data streams. Some of the other data stream processing systems also support stateful continuous queries (i.e. subscriptions). The examples of these systems are: NiagaraCQ [58], Aurora [1], TelegraphCQ [56] and STREAM [140]. NiagaraCQ [58] is a large-scale data stream processing system that supports timer-based and change-based continuous queries. This system achieves scalability by grouping query plans using common expression signatures, i.e. it utilizes the fact that many different queries have the same syntax structure, but different constants. Aurora [1], TelegraphCQ [56] and STREAM [140] are general-purpose data stream processing systems with support for stateful continuous queries. These systems support time-based windows and achieve scalability by computing aggregate values over these windows. We refer a reader interested in data stream processing over sliding windows to check the following references [97, 5, 52].

# CHAPTER 5

## Centralized Publish/Subscribe Systems

In this chapter we focus on centralized publish/subscribe systems. The architecture of a centralized publish/subscribe system is a client-server architecture in which all clients are connected to a centralized publish/subscribe service, as shown in Figure 5.1. In some centralized publish/subscribe systems clients can directly communicate among each other without the mediation of a publish/subscribe service. This is possible since publish/subscribe systems are usually built upon an application-level overlay network. In other words, the clients are not physically connected to a publish/subscribe service, and can thus communicate among themselves using a transport layer protocol. In this chapter we present three commonly used routing strategies for centralized Boolean publish/subscribe systems. These three routing strategies differ among each other according to the location where the matching between publications and subscriptions happens: 1) at the subscriber side, 2) at the publisher side or 3) at the publish/subscribe service side. As the main contribution of this chapter, we explain how the routing strategies can be adapted to centralized top-k/w publish/subscribe systems. This is not a trivial task since Boolean and top-k/w subscriptions are conceptually very different, as explained in Chapter 4. For each of the routing strategies, we show the locations of top-k/w processors and recent buffers in the system, and also explain how subscriptions and publications are routed from clients to the processors. As we will see, two of these three routing strategies are well suited for centralized top-k/w publish/subscribe systems since they do not introduce any additional communication overhead in centralized top-k/w systems when compared to the equivalent centralized Boolean systems. Finally, for all three routing strategies, we experimentally evaluate and compare the performances of Boolean and top-k/w centralized publish/subscribe systems.

The rest of this chapter is organized as follows. In Section 5.1 we present three commonly used routing strategies for centralized Boolean publish/subscribe systems. We adapt these three routing strategies to centralized top-k/w publish/subscribe systems in Section 5.2. In Section 5.3 we experimentally evaluate the performance of Boolean and top-k/w centralized publish/subscribe systems. Finally, we conclude this chapter with an overview of the related work in Section 5.4.

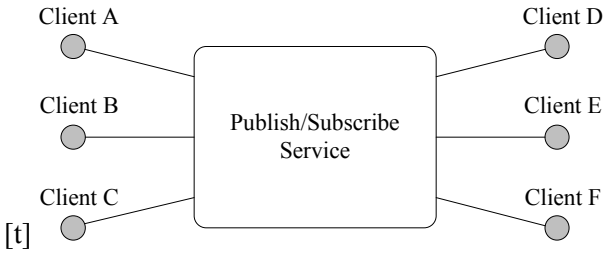



Figure 5.1: Centralized publish/subscribe system architecture.

## 5.1 Routing Strategies in Centralized Boolean Publish/Subscribe Systems

In this section we present three different routing strategies for centralized Boolean publish/subscribe systems (Boolean systems). These three routing strategies differ among each other according to the location where the matching between publications and subscriptions happens: 1) at the subscriber side, 2) at the publisher side, or 3) at the publish/subscribe service side. The publish/subscribe service has a different role in different routing strategies. In the first strategy it acts as a representative of publishers which distributes their publications among other clients. In the second strategy it acts as a representative of subscribers which delivers their subscriptions to other clients. In the third strategy it accepts publications and subscriptions from clients, performs matching and delivers matching publications to their subscribers.

### 5.1.1 Publication Flooding

In a centralized Boolean system with publication flooding, subscriptions are stored at subscribers and therefore none are propagated through the system. When a new publication is published, its publisher forwards it to the publish/subscribe service, which then notifies all other clients in the system about this publication. In this routing strategy, each client performs the matching between a published publication and its own subscriptions. Obviously, this strategy is best used in situations where the majority of clients are interested in each published publication. However, if the number of publishers is too large, the publish/subscribe service will be the bottleneck of the whole system, because it would not be able to distribute all published publications.

**Example 5.1** (Centralized Boolean publish/subscribe system with publication flooding)**.** In Figure 5.2 we can see an example sequence of events in a centralized Boolean system with publication flooding. This system consists of a publish/subscribe service and 6 clients. In this example, client E first subscribes to publications which have $\{x > 10\}$, then client D subscribes to publications for which $\{x > 15\}$, and finally, client A publishes a publication which has $\{x = 20\}$. We can see that subscriptions of clients E and D are not propagated through the system since they are stored at their client nodes. Client A publishes publication $\{x = 20\}$ by forwarding it to the publish/subscribe service, which then notifies all other clients in the system. When a client receives this publication, it performs the matching between the publication and its own subscriptions. As we can see, the publication of client A is matching for subscriptions of clients E and D.

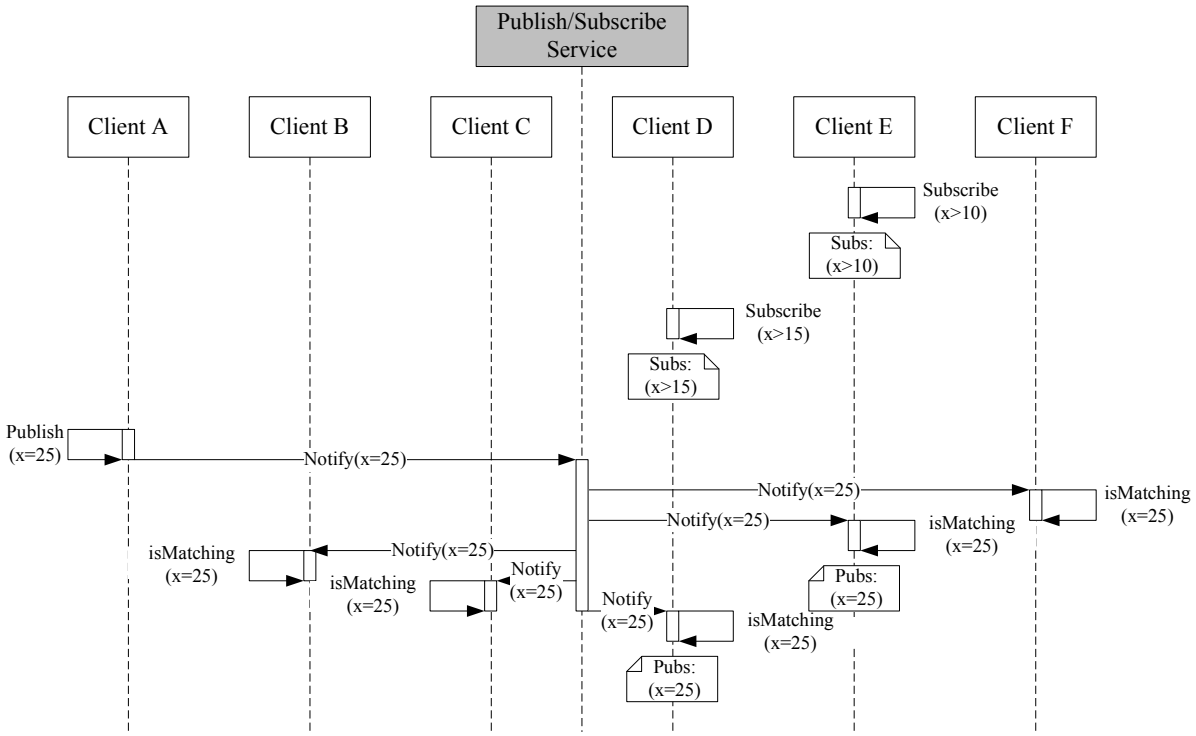



Figure 5.2: Sequence of events in a centralized Boolean system with publication flooding.

The role of a publish/subscribe service in this routing strategy is analogous with the role of an Ethernet hub which connects several Ethernet devices among themselves. The hub receives any packet entering any port and broadcasts it out on all other ports. In this situation, the Ethernet devices are responsible for filtering incoming packets. When the number of devices becomes too large, the overall performance will decrease due to frequent packet collisions on the Ethernet medium.

### 5.1.2 Subscription Flooding

In a centralized Boolean system with subscription flooding, the publish/subscribe service accepts subscription updates (i.e. activations and cancellations) from all clients in the system. When a subscription is activated or canceled, the subscriber forwards such information to the publish/subscribe service, which then informs all other clients in the system about subscription update. As each client stores all active subscriptions in the system, we can use subscription covering and (perfect) merging at the subscriber side to reduce the number of forwarded (i.e. stored) subscriptions in the system and also at the publisher side to improve performance of the matching implementation. In this routing strategy, publishers perform the matching between publications they publish and all active subscriptions in the system. Besides that, they are also responsible to directly deliver (i.e. using a network layer protocol) matching publications to subscribers. This strategy is best suited for sharing the processing load (due to the matching) in situations where the majority of clients are also publishers. Obviously, if the frequency of subscription updates is too large, the publish/subscribe service will become the bottleneck of the whole system, because it will

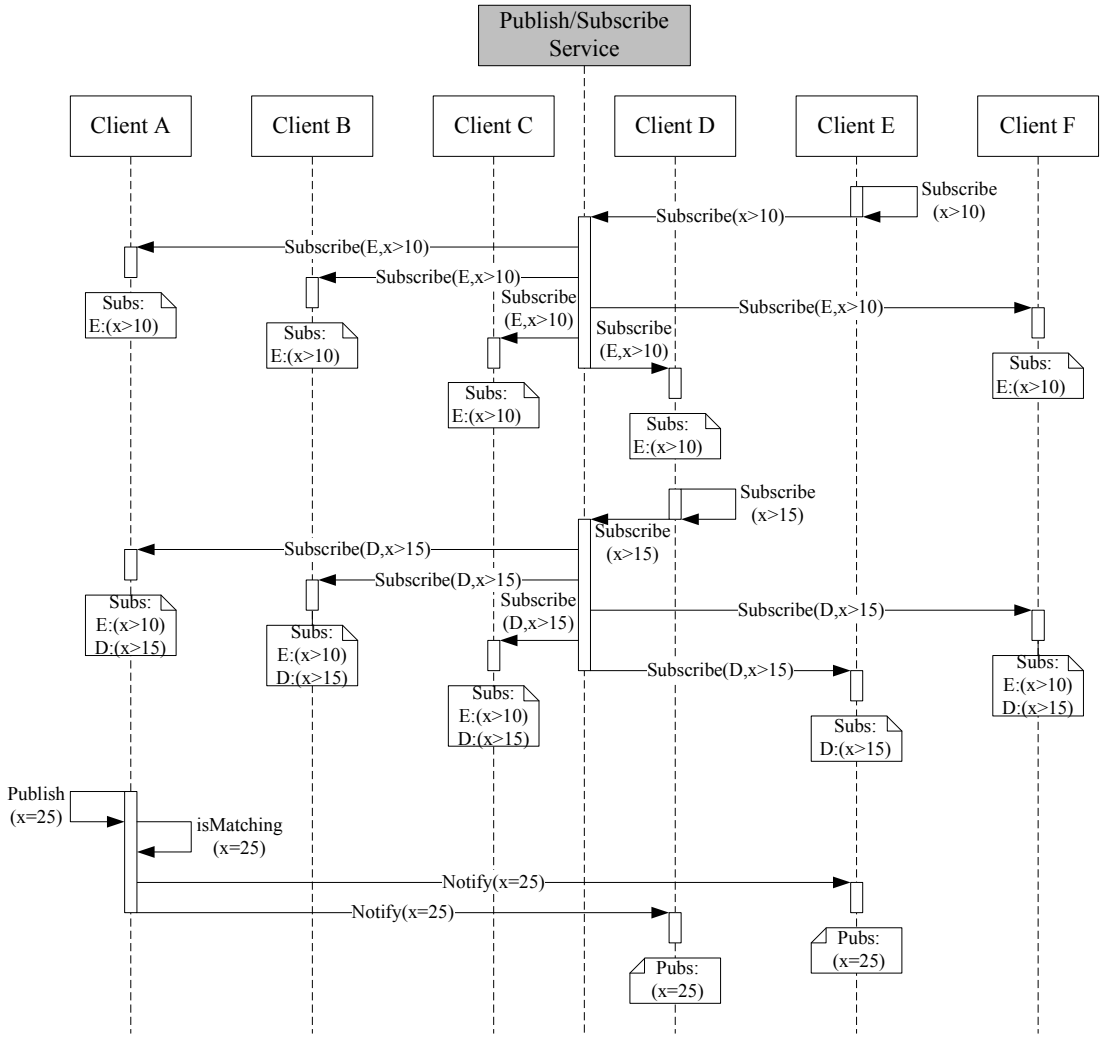



Figure 5.3: Sequence of events in a centralized Boolean system with subscription flooding.

not be able to inform all clients about subscription updates.

**Example 5.2** (Centralized Boolean publish/subscribe system with subscription flooding)**.** Figure 5.3 shows an example sequence of events in a centralized Boolean system with subscription flooding. Similar to Example 5.1, this system consists of a publish/subscribe service and 6 clients where client E first subscribes to publications which have $\{x > 10\}$, then client D subscribes to publications for which $\{x > 15\}$, and finally, client A publishes a publication which has $\{x = 20\}$. We can see that clients E and D forward their subscriptions to the publish/subscribe service, which then informs all other clients in the system about these subscriptions. Client A publishes a publication $\{x = 20\}$ by matching it to the active subscriptions in the system and then notifies clients E and D whose subscriptions are satisfied with this publication.

The role of a publish/subscribe service in this strategy is similar to the role of a service that announces calls for papers by sending e-mails. When a scientist has finished writing a paper, he/she reads the

received announcements, selects one of the calls and submits the paper for review. However, differently from this real-world example, in publish/subscribe systems there is usually more than one subscriber notified about a publication.

The filtering of uninteresting publications at the publisher side is an important characteristic and a big advantage of this routing strategy. However, besides already mentioned scalability issues, three problems arise when this routing strategy is used. The first problem is related to the fact that a publisher and a subscriber have to actively participate in the interaction, which is contradictory to the publish/subscribe communication paradigm [83]. The second problem is a consequence of the first problem and is related to the anonymity, because subscribers and publishers do not remain anonymous with respect to each other. Finally, the third problem is related to trust since the matching happens at the publisher side, which is not under the control of either subscribers or publish/subscribe service and can easily be abused. In other words, subscribers have to trust publishers that incoming publications are really matching their subscriptions.

### 5.1.3 Selective Routing

In a centralized Boolean system with selective routing, the publish/subscribe service accepts publications and subscription updates (i.e. activations and cancellations) from all clients in the system. It stores all active subscriptions in the system, and performs matching between incoming publications and these subscriptions. Additionally, the publish/subscribe service is also responsible for notifying subscribers about publications matching their subscriptions. This strategy is best used in situations when we want to reduce the communication overhead in the system and processing overhead at the client side. This is achieved by avoiding the passing of redundant information to clients. Obviously, if the number of clients is too large, the publish/subscribe service would not be able to process (i.e. perform matching for) all published publications and thus will be the bottleneck of the whole system.

**Example 5.3** (Centralized Boolean publish/subscribe system with selective routing)**.** Figure 5.4 shows an example sequence of events in a centralized Boolean system with selective routing. Similar to Examples 5.1 and 5.2, this system consists of a publish/subscribe service and 6 clients where client E first subscribes to publications which have $\{x > 10\}$, then client D subscribes to publications for which $\{x > 15\}$, and finally, client A publishes a publication which has $\{x = 20\}$. We can see that subscriptions of clients E and D are forwarded by these clients to the publish/subscribe service, which stores them in its memory. Client A publishes a publication $\{x = 20\}$ by forwarding it to the publish/subscribe service. This service then performs the matching between this publication and stored subscriptions and then notifies clients E and D since their subscriptions are satisfied with the publication.

When this routing strategy is used in a centralized Boolean system, the publish/subscribe service acts as a publication filter. It forwards only matching publications to subscribers, and ignores (does not pass) other publications it receives from publishers. In this way, the communication overhead in the system is reduced as much as possible since the system is not flooded neither with publications nor subscriptions and only matching publications are forwarded to subscribers.

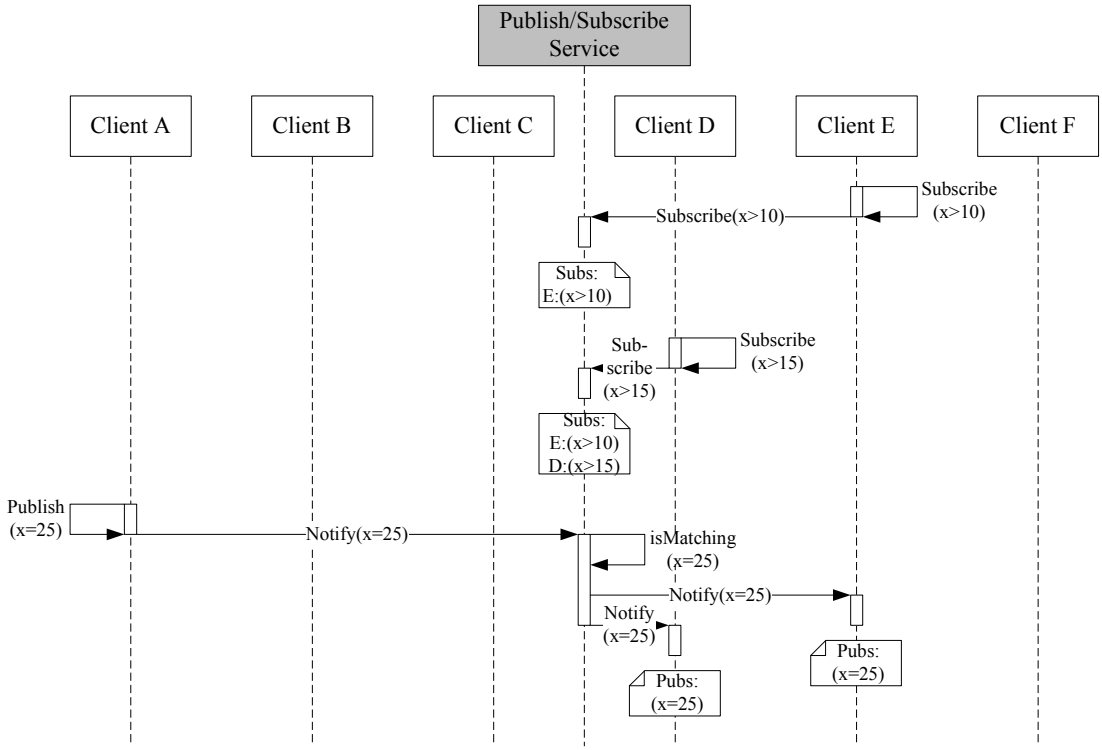



Figure 5.4: Sequence of events in a centralized Boolean system with selective routing.

## 5.2 Routing Strategies in Centralized Top-k/w Publish/Subscribe Systems

In the previous section we presented three routing strategies for centralized Boolean publish/subscribe systems. In this section we adapt these three routing strategies for centralized top-k/w publish/subscribe systems (centralized top-k/w systems). For each of the strategies, we explain in detail differences when compared to the original routing strategy, identify optimal locations of top-k/w processors and recent buffers in system, and explain how subscriptions and publications are routed from clients to these processors.

### 5.2.1 Publication Flooding

Figure 5.5 shows entities and their roles in a centralized top-k/w system with publication flooding. As we can see, each subscriber has a heavyweight top-k/w processor and recent buffer to process the stream of incoming publications for its own subscriptions. This processor knows the current thresholds of subscriptions it stores in memory, and can thus without any doubt detect matching publications for these subscriptions. In this routing strategy we do not need to store proxy subscriptions in the network since the matching is performed at the subscriber side. We conclude that this routing strategy is well suited for centralized top-k/w systems because it does not introduce any additional communication overhead when compared to centralized Boolean systems.

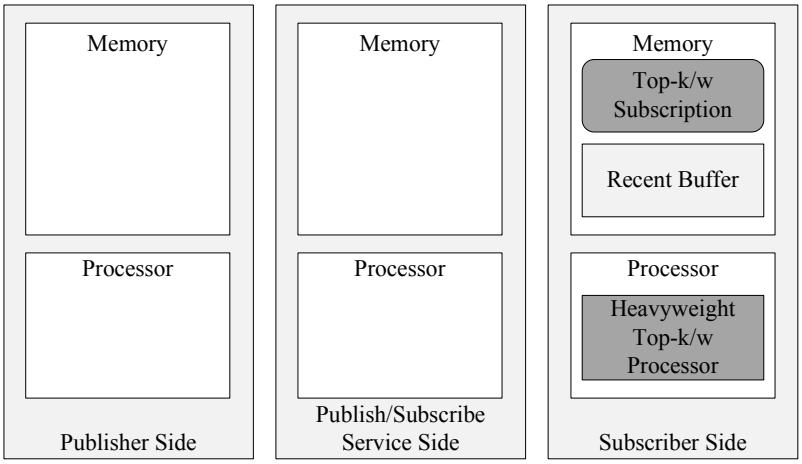



Figure 5.5: Centralized top-k/w publish/subscribe system with publication flooding.

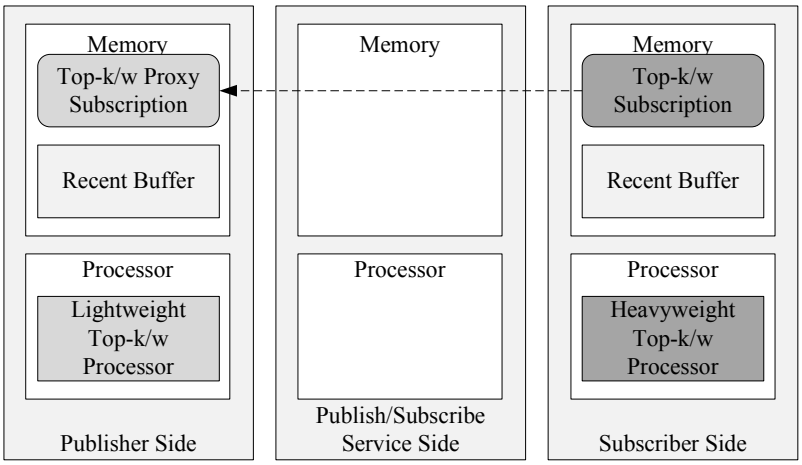



Figure 5.6: Centralized top-k/w publish/subscribe system with subscription flooding.

### 5.2.2 Subscription Flooding

Figure 5.6 shows entities and their roles in a centralized top-k/w system with subscription flooding. As we can see, each publisher node has a lightweight top-k/w processor and recent buffer to process the stream of its own publications for proxy subscriptions of all active subscriptions in the system. The matching between a published publication and stored proxy subscriptions is performed twice[1] at the publisher side. The first matching is performed immediately after publishing, whereas the second is performed immediately after dropping this publication from the recent buffer. After the first matching, the publisher directly forwards (i.e. using a network layer protocol) the publication to subscribers with satisfied subscriptions. However, after the second matching it directly forwards this publications only to those subscribers whose subscriptions are now satisfied, but were not during the first matching. Each subscriber must have a heavyweight top-k/w processor and recent buffer to process the stream of received publications for its own subscriptions. When a subscription threshold changes, the subscriber forwards the new value to the publish/subscribe service, which then informs all other clients in the system to update their

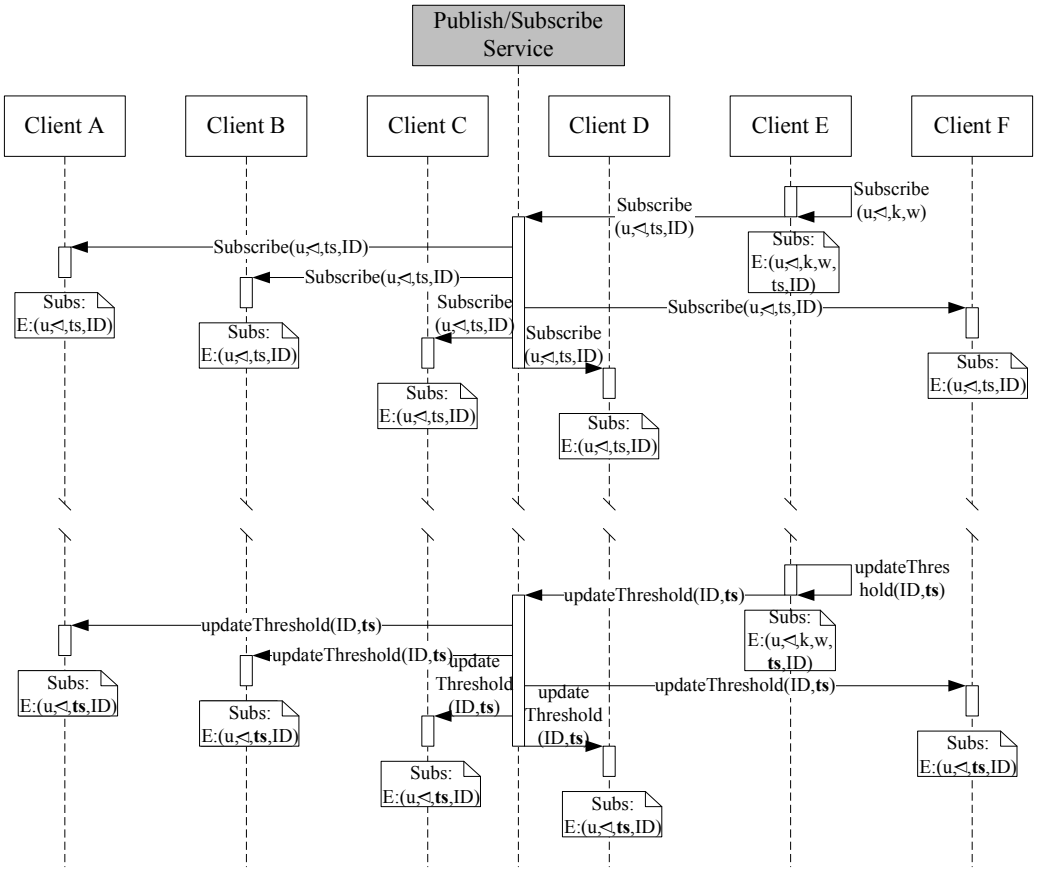



Figure 5.7: Sequence of events in a centralized top-k/w system with subscription flooding.

proxy subscriptions. We conclude that this routing strategy can be implemented for centralized top-k/w systems, but is not well suited for them since the synchronization of subscription thresholds introduces a large communication overhead when compared to the equivalent centralized Boolean systems.

It is very important to notice that a more critical situation is when a subscription subspace of interest expands than the opposite situation since the subscriber could miss some matching publications if this information is not instantly propagated through the system. In the latter situation the subscriber could be notified about some publications which are no more matching, but as in practice these publications can be filtered at the subscriber side, this is a less critical situation.

**Example 5.4** (Centralized top-k/w publish/subscribe system with subscription flooding)**.** Figure 5.7 shows a sequence of events in a centralized top-k/w system with subscription flooding. This example shows how information about a threshold change spread through this system. Similar to Boolean system examples, this system consists of a publish/subscribe service and 6 clients. We can see that client E first activates its subscription by forwarding it to the publish/subscribe service. After some time, when the threshold of this subscription changes, client E informs the publish/subscribe service about this change,

[1]These two matchings are analogous with the two insertion attempts for SA and RA with filters, which are explained in Chapter 3. In the case of PA, we do not need a recent buffer at the publisher side since the matching is performed only once, immediately after the publishing.

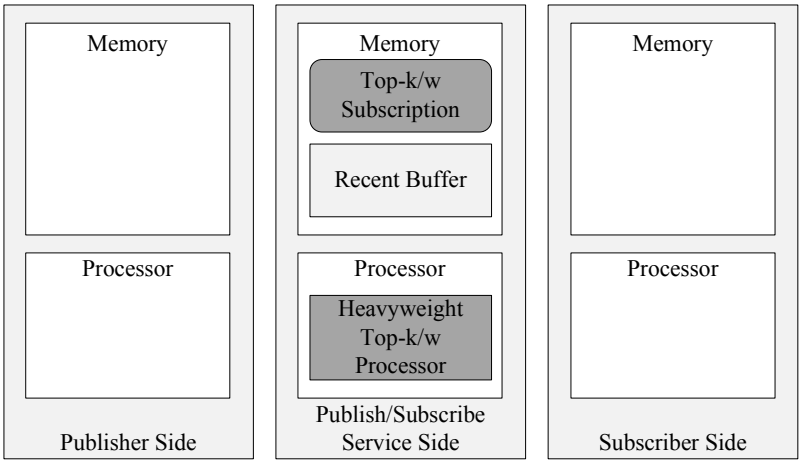



Figure 5.8: Centralized top-k/w publish/subscribe system with selective routing.

which then informs all other clients in the system to update their proxy subscriptions.

### 5.2.3 Selective Routing

Figure 5.8 shows entities and their roles in a centralized top-k/w system with selective routing. As we can see, we need only one heavyweight top-k/w processor and only one recent buffer in the system. Both of them are located at the publish/subscribe service side. This processor knows the current thresholds of all active subscriptions in the system, and thus can without any doubt detect matching publications for them. Since the matching is performed only at the publish/subscribe service side, in this routing strategy we do not need to store proxy subscriptions in network. We conclude that this routing strategy is well suited for centralized top-k/w systems because it does not introduce any additional communication overhead when compared to centralized Boolean systems.

## 5.3 Experimental Evaluation

In this section we present an experimental study comparing three possible centralized implementations of a Boolean and equivalent top-k/w system. For top-k/w subscriptions we employ two algorithms presented in Chapter 3: PA and RAPF. The former algorithm is probabilistic, while the latter is the fastest of the deterministic top-k/w processing algorithms. All algorithms are implemented in Java and experiments were performed on a Linux based PC with 2.13Ghz Intel® Core™2 Duo CPU (with one core disabled) and 2GB of memory.

Similarly as in Chapters 3 and 4, we have selected NN subscriptions (i.e. NN queries) as a use case for our evaluation. Both subscriptions and publications in our experiments are represented as points in a $d$-dimensional Euclidean space. The score of an publication $p$ with respect to subscription $s$ is calculated using the following formula: $u_s(p) = d(p_s, p_p) = [\sum_{i=1}^{d}(v_i - v_i)^2]^{\frac{1}{2}}$, where $p_p = \{v_1, v_2, \ldots, v_d\}$ and $p_s = \{v_1, v_2, \ldots, v_d\}$ are points representing $p$ and $s$, respectively. The score comparator $\overset{s}{\rhd}$ is

Table 5.1: Default values of parameters used in the experimental evaluation of centralized publish/subscribe systems.

| Parameter | Symbol | Value |
|---|---|---|
| Number of publications | $N$ | $10^6$ |
| Number of subscriptions | $m$ | 400 |
| Size of recent buffer | $b$ | 2000 |
| Data dimensionality | $d$ | 4 |
| Grid resolution | $\rho$ | 10 |
| RA: pruning coefficient | $\gamma$ | 0.2 |
| PA: probability of error | $\sigma$ | $10^{-3}$ |
| Number of clients | $C$ | 400 |

defined so that lower scores imply higher ranks. Additionally, (ranking) Boolean subscriptions define static thresholds, whereas top-k/w subscriptions define parameters $k$ and $w$.

Please note that for the sake of simplicity, we omit subscripts of subscription parameters in this section, if otherwise not explicitly stated.

We have also used one real and two synthetic datasets in the experimental evaluation. In particular, we used the LUCE deployment data (environmental data collected from large-scale wireless sensor networks within the project SensorScope[2]), and generated uniform and clustered Gaussian data. The LUCE deployment data was preprocessed to extract 4-dimensional points (solar panel current, global current, primary buffer voltage and secondary buffer voltage) and normalized to the values within the interval $[0, 1]$, while the synthetically generated data is also within the same interval.

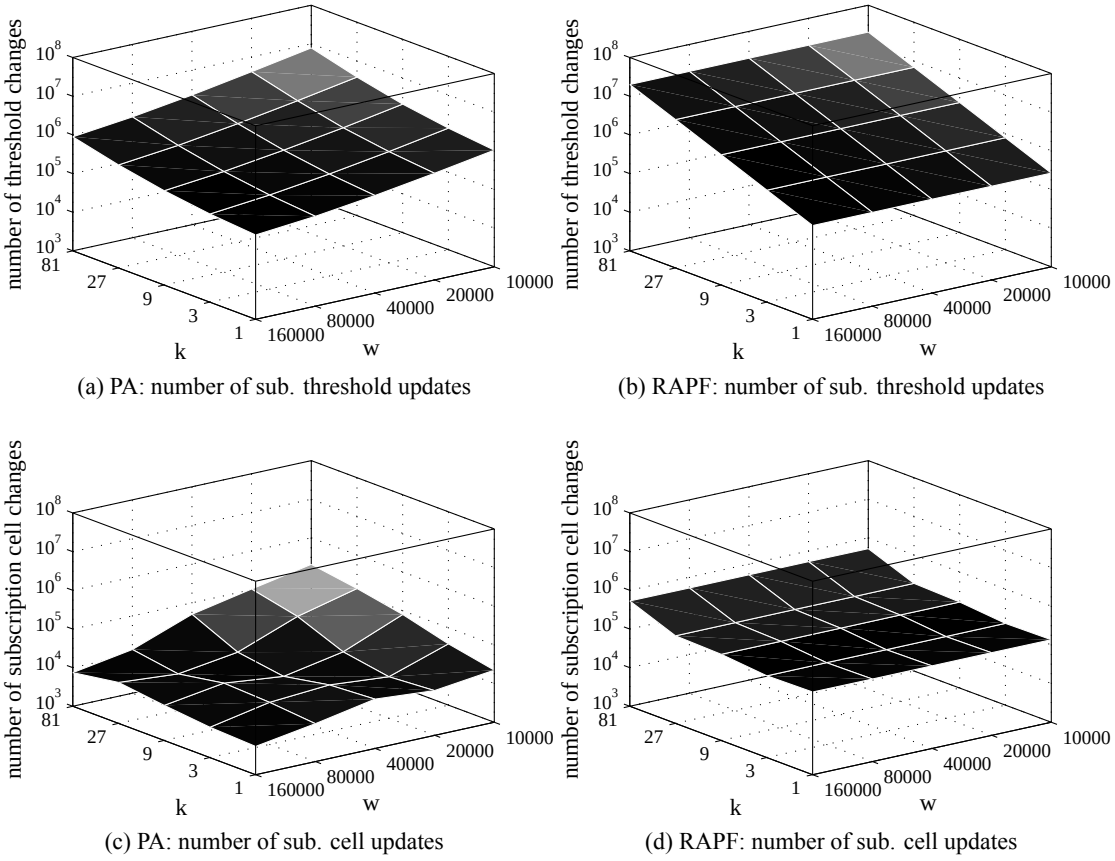
The default scenario used in all experiments is the following: First we generated the set of subscriptions, either by taking a random sample from the LUCE deployment data, or by generating subscriptions using one of the listed distributions. Second we simulated the publishing of publications, either by randomly choosing publications from the LUCE deployment data, or by generating publications using the same distribution as for the previously generated queries. Finally, after $N$ published publications, we analyzed the obtained results. The default simulation parameters used in experiments are specified in Table 5.1. Additionally, we assumed that each client issues a single subscription, i.e. that $m = C$.

In the following we examine the observed number of exchanged messages in the Boolean and top-k/w system. Since the number of exchanged messages is identical when publication flooding is used as the routing strategy, we compare these two systems only for subscription flooding and selective routing strategies. For selective routing, we also compare processing costs of the Boolean system and two top-k/w systems: one utilizing PA and the other utilizing RAPF.

### 5.3.1 Subscription Flooding

In this simulation scenario we compare the number of exchanged messages in the Boolean and top-k/w system when subscription flooding is used as the routing strategy. We used only the uniform dataset for

[2]http://sensorscope.epfl.ch/




(a) PA: number of sub. threshold updates

(b) RAPF: number of sub. threshold updates

(c) PA: number of sub. cell updates

(d) RAPF: number of sub. cell updates

Figure 5.9: Number of subscription threshold and cell updates for different types of top-k/w subscriptions in a centralized top-k/w system with subscription flooding.

this experiment. The number of exchanged messages in the Boolean system is the sum of the following:

- the number of subscribe messages and
- the number of notify messages.

Similarly, the number of exchanged messages in the top-k/w system is the sum of the following:

- the number of subscribe messages,
- the number of notify messages, and
- the number of update (proxy subscription) messages.

The number of subscribe messages is equal to $m \cdot C = 160,000$ since every such message is distributed by the publish/subscribe service to the other clients in system. The number of notify messages is equal to the number of matching publications, and this number (per Boolean and top-k/w subscription) is already shown in Figure 4.8. Therefore, up to this point, only the number of update messages in the top-k/w system is unknown.

We analyze two approaches for informing other clients about a subscription update. The first approach informs about every subscription update, while the second divides the Euclidean space to cells of equal size (regular grid), and informs only when a subscription subspace of interest expands to new cells, or contracts from old cells. The numbers of subscription threshold and cell changes are shown in Figure 5.9. To get the total number of update messages, we have to multiply the number of updates with the number of clients in the system. We can see that the number of updates is much lower in the case of the latter approach, which can be used for top-k/w systems, while the former should be avoided due the frequent flooding with threshold update messages. Please note that each change in subscription cells may refer to one or more cells and thus the corresponding messages are larger than threshold update messages of the former approach, which carry only a subscription ID and new threshold value.

### 5.3.2 Selective Routing

In this simulation scenario we compare the number of exchanged messages and processing cost in the Boolean system and top-k/w system when the selective routing strategy is used. The number of exchanged messages for both systems is the sum of the following:

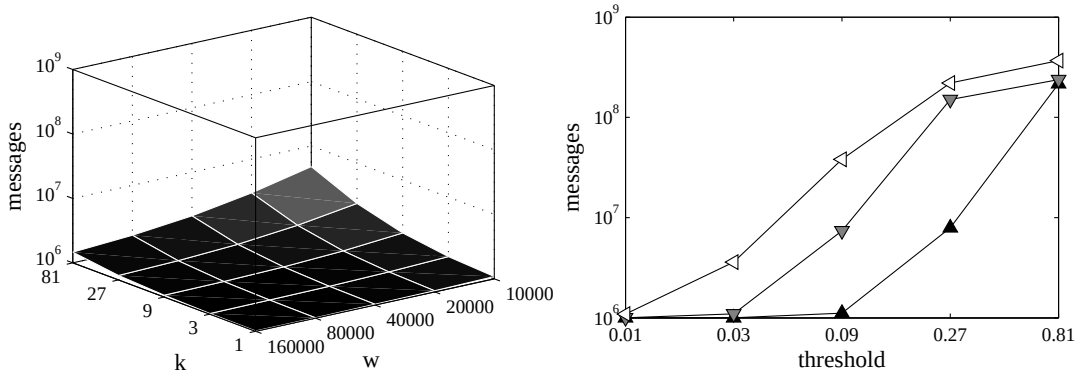
- the number of subscribe messages,
- the number of publish messages and
- the number of notify messages.

The number of subscribe messages is equal to $m = 400$, the number of publish messages is equal to $N = 1,000,000$, and the number of notify messages is equal to the number of matching publications which is already shown (per subscription) in Figure 4.8. Figure 5.10 shows the total number of exchanged messages for the Boolean and top-k/w system. We see that this routing strategy can significantly reduce message overhead in a top-k/w system when compared to the equivalent Boolean system in which subscription matching functions (i.e. Boolean subscription thresholds in our case) are inadequately selected for the actual distribution of publications.

In the second experiment we examine the processing cost at the publish/subscribe service side in the Boolean system and two top-k/w systems: one utilizing PA and the other utilizing RAPF. The processing cost is expressed as runtime for different datasets, both with and without query indexing using a regular grid.

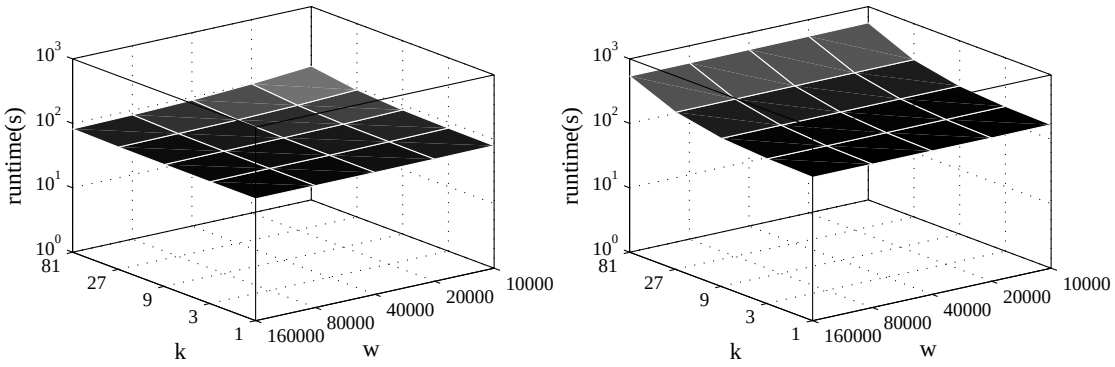
Figure 5.11 shows the processing costs without query indexing. We can see that processing costs for the Boolean system and top-k/w system with PA are very similar, while the top-k/w system with RAPF has much worse processing performance. For the former two systems, the processing cost is similar since it is mainly due to the publication score calculation.

Figure 5.12 shows the processing costs with query indexing. As expected, the top-k/w system with PA has lower processing cost than the top-k/w system with RAPF. For the uniform dataset and ideally selected Boolean subscription thresholds, the Boolean system has lower processing cost than the top-k/w

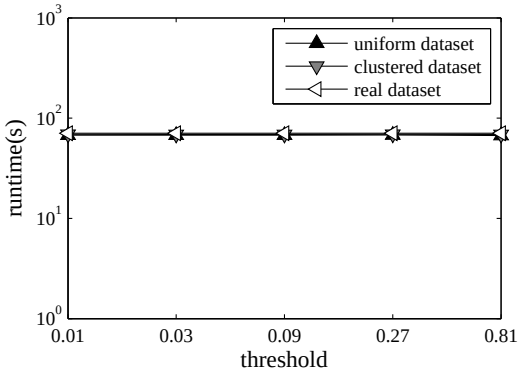



(a) Top-k/w system (all datasets) (b) Boolean system

Figure 5.10: Total number of exchanged messages in a centralized Boolean and top-k/w system with selective routing.



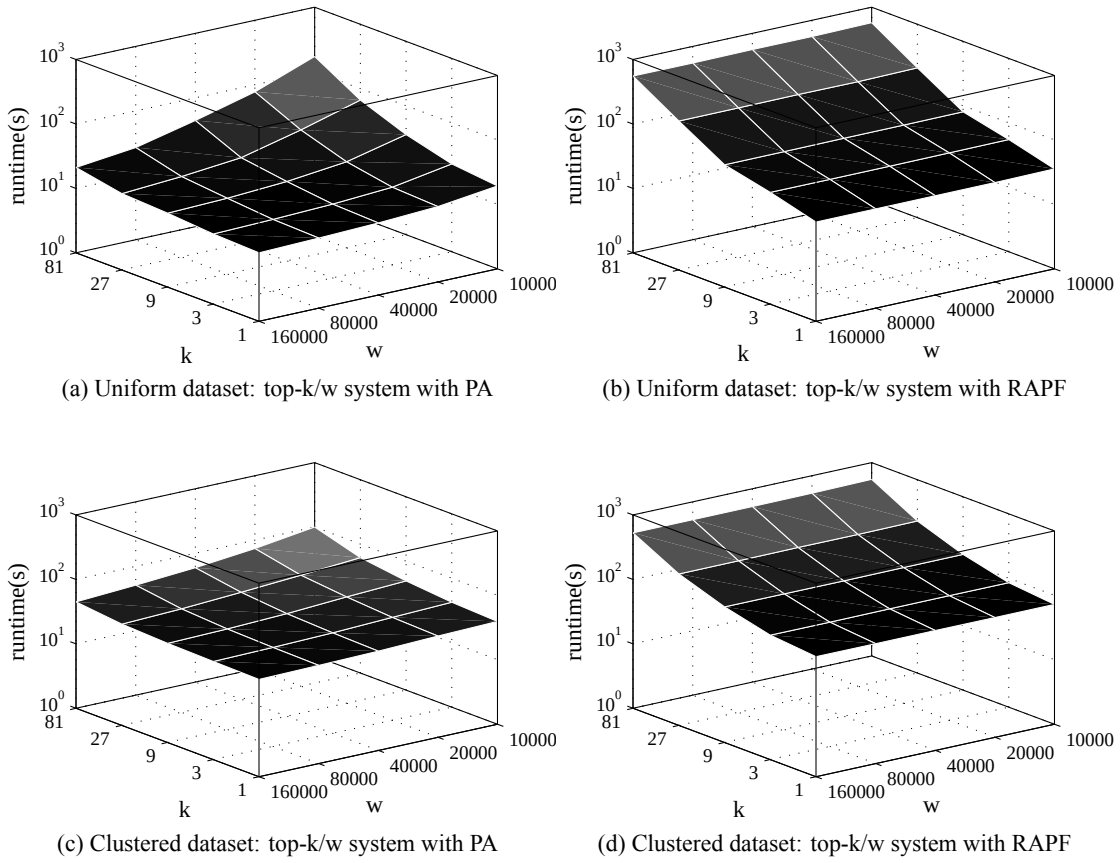

(a) Top-k/w system with PA (b) Top-k/w system with RAPF




(c) Boolean system

Figure 5.11: Processing cost without query indexing in a centralized Boolean and top-k/w system with selective routing.

system with PA. However, if the subscription threshold is inadequately selected, we can see that the top-




(a) Uniform dataset: top-k/w system with PA

(b) Uniform dataset: top-k/w system with RAPF

(c) Clustered dataset: top-k/w system with PA

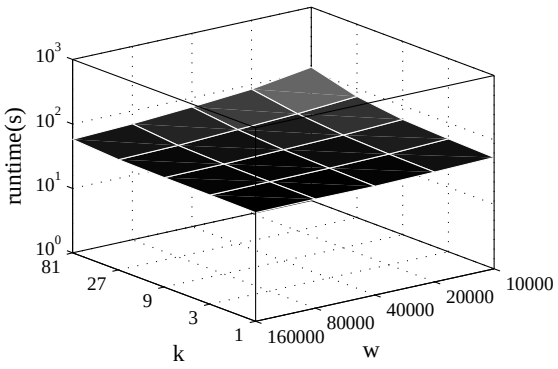
(d) Clustered dataset: top-k/w system with RAPF

Figure 5.12: [First part] Processing cost with query indexing for different datasets in a centralized Boolean and top-k/w system with selective routing.

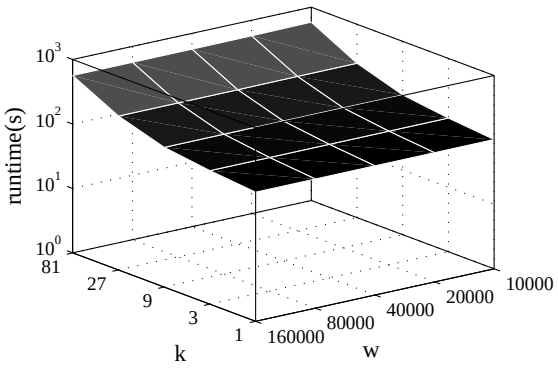
k/w system has equally good or even better processing performance than the Boolean system. For all systems, the processing performance is generally better for the uniform than other two datasets, which is expected since we are using the regular grid. Surprisingly, for the uniform dataset, larger values of the parameter k and smaller of the parameter w, the top-k/w system behaves worse than for the other two datasets. Our explanation for this behavior is in the increased values of subscription thresholds in the case of uniform dataset when compared to the other datasets, which requires frequent grid reconfigurations. Please recall that the average threshold values are already shown in Figure 4.6.

### 5.3.3 Conclusion

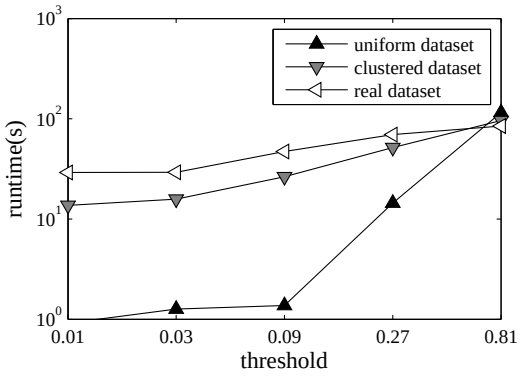
The presented experimental evaluation has shown that subscription flooding as a routing strategy in centralized publish/subscribe systems is not well suited for top-k/w systems since it frequently floods clients with update (proxy subscription) messages. Additionally, it has experimentally demonstrated that centralized top-k/w systems use less resources and generate less messages than the equivalent Boolean systems in which subscriptions are inadequately defined.




(e) Real dataset: top-k/w system with PA




(f) Real dataset: top-k/w system with RAPF




(g) All datasets: Boolean system

Figure 5.12: [Second part] Processing cost with query indexing for different datasets in a centralized Boolean and top-k/w system with selective routing.

## 5.4 Related Work

Centralized publish/subscribe systems are very popular in practice, mainly due to the fact that they offer all the advantages of publish/subscribe messaging paradigm, while at the same time are simple to build and very easy to maintain. The general ideas and essential features of centralized publish/subscribe systems are of widely applicable and thus they have attracted a lot of attention from both the research and industry communities in the last 20 years. However, because of the lack of scalability, these systems cannot be used to deploy large-scale publish/subscribe solutions. In this section we present an overview of centralized publish/subscribe systems in which special attention will be given to the used routing strategies.

### 5.4.1 Centralized Publish/Subscribe Systems

Publication flooding as a routing strategy was used in the first centralized publish/subscribe [155] and similar systems [113, 3, 185]. These systems used LAN or air as a medium for disseminating information to subscribers, which then locally filtered out information that did not match their interest. In this case, the medium acts as a publish/subscribe service, which is a very elegant and simple solution, especially

for small-scale publish/subscribe systems which exploit LAN broadcast capability. However, this routing strategy should be avoided for large-scale publish/subscribe systems since it would generate a high traffic load. This strategy is also applied in some more recent distributed publish/subscribe systems, e.g. Gryphoon [24], JEDI [69] and TERA [19].

The opposite routing strategy to the former is subscription flooding. This strategy was also used in some early centralized publish/subscribe systems such as Elvin3 [187]. In Elvin3 this strategy was called quenching, an is particularly effective for reducing network traffic in situations when a large number of publications do not match any of the subscriptions. Some of the more recent and distributed publish/subscribe systems also apply this strategy, e.g. Gryphoon [24], RUBDES [184] and MEDYM [41].

The most popular routing strategy for centralized publish/subscribe systems is selective routing. This strategy has become a de facto standard for centralized publish/subscribe systems. It is often considered in literature, e.g. [120, 203, 161, 85, 168], and used in specifications, e.g. Corba Event Service [153], Corba Notification Service [154] and Java Message Service [107]. Additionally, many open source and proprietary providers currently offer messaging frameworks or Enterprise Service Bus (ESB) solutions which include publish/subscribe messaging with selective routing as their standard part. The examples are Apache ActiveMQ[3], Fiorano Enterprise Service Bus[4], IBM WebSphere Enterprise Service Bus[5], JBoss-ESB[6], Microsoft BizTalk Server[7], Oracle Enterprise Service Bus[8], Progress Sonic ESB[9], Sun GlassFish Message Queue[10], TIBCO Enterprise Message Service[11], etc.

### 5.4.2 Matching Algorithms in Publish/Subscribe Systems

The matching between incoming publications and a large number of stored subscriptions is the most computationally intensive task performed by the publish/subscribe service in a centralized publish/subscribe system with selective routing. In this paragraph we give a brief survey of matching algorithms in publish/subscribe systems. According to [179], these algorithms can be divided in two groups, namely predicate indexing based algorithms and testing network based algorithms. Both groups of algorithms assume that a subscription is a conjunction of predicates, where each predicate is a triple of the form attribute-operator-element. The former group of algorithms [106, 202, 162, 161, 85, 74, 48] preprocess subscriptions and place (some or all) their predicates in an index structure. Upon a new publishing, they perform matching in two phases: 1) evaluate predicates and 2) find satisfied subscriptions. On the other hand, the latter group of algorithms [99, 6, 39, 125] preprocess subscriptions by building a data structure whose internal and leaf nodes are subscription predicates and subscriptions, respectively. The matching is done by traversing this data structure from the root and through the matched internal nodes to the leaves. Finally,

[3] http://activemq.apache.org/
[4] http://www.fiorano.com/products/fesb/products_fioranoesb.php
[5] http://www-01.ibm.com/software/integration/wsesb/
[6] http://www.jboss.org/jbossesb
[7] http://www.microsoft.com/biztalk/en/us/default.aspx
[8] http://www.oracle.com/appserver/esb.html
[9] http://web.progress.com/en/sonic/index.html
[10] http://www.sun.com/software/products/message_queue/index.xml
[11] http://www.tibco.com/software/messaging/enterprise-message-service/default.jsp

the most recent work [87] in this field leverages current chip multi-processors to increase throughput and to reduce the matching time by parallelizing the matching algorithm presented in [85].

It is important to notice that these matching algorithms are compatible with top-k/w subscriptions. As we know from Chapter 3, each top-k/w subscription may additionally define a minimal score for matching publications. Therefore, upon the publishing of a new publication, we can use the above specified algorithms to avoid score calculation for those subscriptions for which a score of this publication would be worse than their minimal score. In other words, these algorithms can also be used for subscription indexing.

# CHAPTER 6

## Distributed Publish/Subscribe Systems

In this chapter we focus on distributed publish/subscribe systems. The architecture of such systems is distributed and composed of clients and broker servers that actively participate in the system without the need for central coordination as in the case of centralized publish/subscribe systems. In distributed publish/subscribe systems, all clients (i.e. brokers) coordinate their actions to achieve a common goal, and that is to provide the publish/subscribe service both to themselves and to other clients in the system. We begin this chapter by presenting the architectural model of distributed publish/subscribe systems from [23, 22]. This model groups the commonly used routing strategies in distributed publish/subscribe systems to three major groups, where each of them includes two commonly used strategies. In this chapter we present these six commonly used routing strategies in detail. As the main contribution of this chapter, we explain how these routing strategies can be adapted to distributed top-k/w publish/subscribe systems. Again, this is not a trivial task since Boolean and top-k/w subscriptions are conceptually very different, as explained in Chapter 4. For each of the routing strategies, we show locations of top-k/w processors and recent buffers, and also explain how subscriptions and publications are routed from clients to these processors. As we will see, four of six routing strategies are well suited for distributed top-k/w publish/subscribe systems since they introduce marginal communication overhead when compared to the equivalent distributed Boolean systems. In this chapter, we present D-ZaLaPS, a distributed top-k/w publish/subscribe system with rendezvous routing[1] supporting distance scoring functions which is built on top of Content Addressable Network (CAN) [178]. D-ZaLaPS supports both Boolean and top-k/w subscriptions with distance scoring (i.e. matching) functions. In this chapter we also experimentally evaluate the performance of D-ZaLaPS for both types of subscriptions.

The rest of this chapter is organized as follows. In Section 6.2 we present three commonly used routing strategies for distributed Boolean publish/subscribe systems. We adapt these six routing strategies

[1] The rendezvous routing is one of the four well suited routing strategies for top-k/w systems.

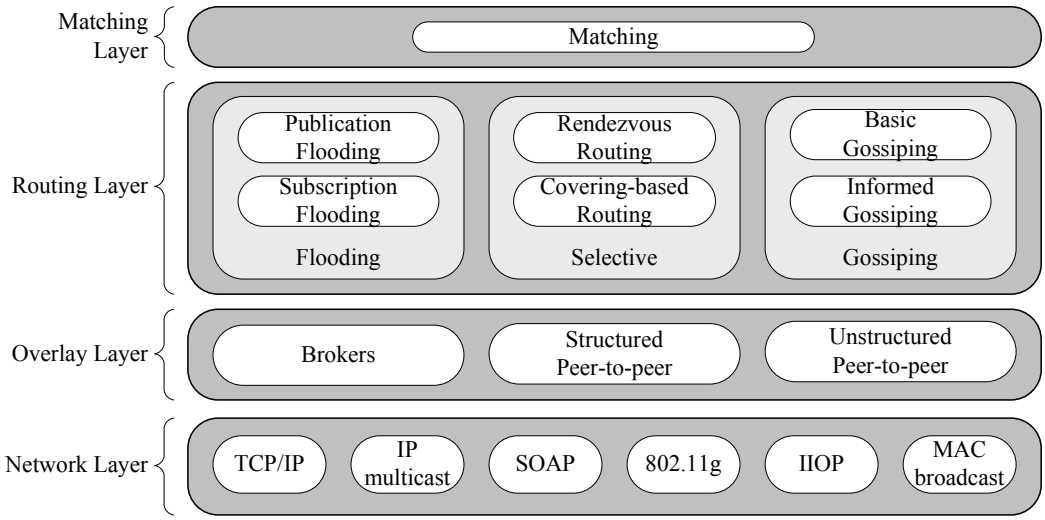



Figure 6.1: Publish/subscribe architectural model from [23, 22]

to distributed top-k/w publish/subscribe systems in Section 6.3. We present D-ZaLaPS in Section 6.4 and experimentally evaluate it in Section 6.5. Finally, we conclude this chapter with an overview of related work in Section 6.6.

## 6.1 Architectural Model of Distributed Publish/Subscribe Systems

In this section we briefly describe the publish/subscribe architectural model [23, 22] we consider in this chapter. This model is shown in Figure 6.1, and consists of four logical layers: network layer, overlay layer, routing layer and matching layer. This model primarily shows the architecture of distributed publish/subscribe systems since the architecture of centralized publish/subscribe systems is much simpler as explained in Chapter 5.

The network layer is responsible for communication between network nodes (i.e. clients or brokers) in a publish/subscribe system. Usually, the Internet protocol suite (TCP/IP) is used for this purpose, but alternative approaches like network-layer multicast, mobile ad-hoc networks and sensor networks are also applied in practice. Further analysis of the protocols in this layer is out of scope of this thesis, and thus we refer an interested reader to [23, 22] for more details about this layer.

In practice, a distributed publish/subscribe system is built on top of an application-layer overlay network. Three most widely-used overlay networks in distributed publish/subscribe systems are broker overlay, peer-to-peer structured overlay and peer-to-peer unstructured overlay. The broker network is a manually managed and mainly static network of *brokers* that act as servers of publish/subscribe service to clients. Brokers exchange messages containing subscriptions and publications and share the service load among themselves. This overlay network is most used in practice due to its simplicity. The structured peer-to-peer network is a self-organized structured overlay network of equal *peer* nodes that simultaneously act as clients and servers to other nodes in this network. In a structured peer-to-peer overlay network, an attribute or vector space is mapped to a virtual address space shared between the peers and

then parts of this virtual space are further mapped to active peers in this network. An unstructured peer-to-peer network is a self-organized overlay network of peers that has a small diameter, does not have a defined structure and where only locally available information is used for the network maintenance. Due to the last fact, this overlay network is much better suited for systems with highly dynamic clients than the structured peer-to-peer overlay network.

The core mechanism behind a distributed publish/subscribe service is routing publications and subscriptions between nodes in an overlay network. The main issue with routing algorithms is their scalability, which requires careful balancing of the number of propagated messages in an overlay network and the amount of stored routing information at overlay nodes. The following three groups of routing algorithms are used in distributed publish/subscribe systems: flooding, selective routing and gossiping. Flooding can be implemented as either subscription or publication flooding. This type of routing causes a large message overhead since the overlay network is frequently flooded with messages. Selective routing is further divided to covering-based[2]and rendezvous routing. This type of routing tries to reduce the message overhead in an overlay network as much as possible. Selective filtering is based on the following two facts: 1) subscribers are usually interested in just a portion of published publications and 2) the interests of different subscribers usually overlap. When this is not true selective filtering will lead to flooding of an overlay network with either subscriptions or publications. Finally, the gossiping algorithms used with unstructured peer-to-peer overlay networks can further be categorized as basic gossiping, which is purely probabilistic, and informed gossiping, which is partially probabilistic and partially deterministic.

The matching of publications to subscriptions is discussed in detail in Chapters 1, 2 and 5 and is thus not further elaborated it in this section. In practice, the routing of messages (i.e. publications and subscriptions) in a distributed publish/subscribe systems is very much related to the process of matching between publications and subscriptions. As a consequence, it is hard to draw a sharp boundary between the matching and routing layers. For this reason, in the following sections we separately analyze each of the six routing strategies for Boolean and top-k/w distributed publish/subscribe systems.

## 6.2 Routing Strategies in Distributed Boolean Publish/Subscribe Systems

The referent publish/subscribe architectural model presented in Section 6.1 identifies six commonly used strategies in distributed Boolean publish/subscribe systems (distributed Boolean systems). In this section we analyze these six routing strategies in detail. Depending on the used routing strategy, the matching process between publications and subscriptions in a distributed Boolean systems happens at one of the following locations: 1) the subscriber side, 2) the publisher side or 3) the third side. For each of these six routing strategies we present an example sequence of events assuming the overlay network topology shown in Figure 6.2. This topology consists of six nodes connected in an acyclic simple graph (i.e. tree) and is a generalization of all three overlay networks presented in Section 6.1.

[2]The publish/subscribe architectural model from [23, 22] considers filtering-based instead of covering-based routing. However, covering-based routing is an advanced version of filtering-based routing, and is more often used in practice.

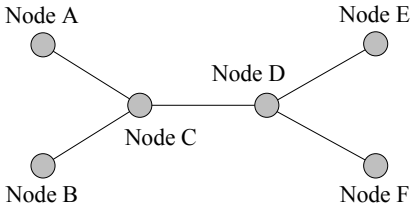



Figure 6.2: Overlay network topology of a distributed publish/subscribe system.

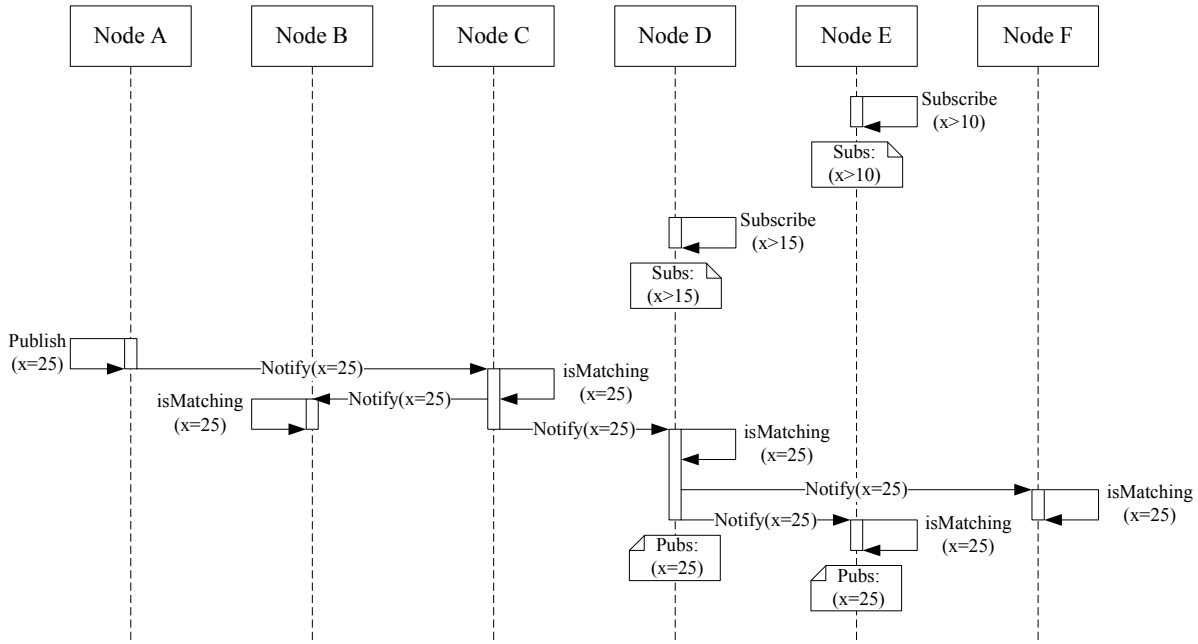



Figure 6.3: Sequence of events in a distributed Boolean system with publication flooding.

### 6.2.1 Publication Flooding

In a distributed Boolean system with publication flooding, subscriptions are stored at subscriber nodes and are therefore non propagated further into the overlay network. When a publication is published, the overlay network is completely flooded with it. Upon receiving a newly published publication, each node forwards this publication to its neighbors, i.e. further into the overlay network, and after that begins the process of matching between this publication and its own subscriptions to check if this publication matches any of them. Obviously, this strategy is best suited in situations when the majority of nodes is interested in most of the published publications. However, if the number of publishers is large and they have high publishing rate, the overlay network will be overloaded with published publications.

**Example 6.1** (Distributed Boolean publish/subscribe system with publication flooding)**.** In Figure 6.3 we can see an example sequence of events in the distributed Boolean system with publication flooding whose topology is shown in Figure 6.2. This sequence is analogous to the examples in Chapter 5 because node E first subscribes to publications that satisfy the condition $\{x > 10\}$, then node D subscribes to publications for which $\{x > 15\}$, and finally, node A publishes a publication $\{x = 20\}$. We can see that

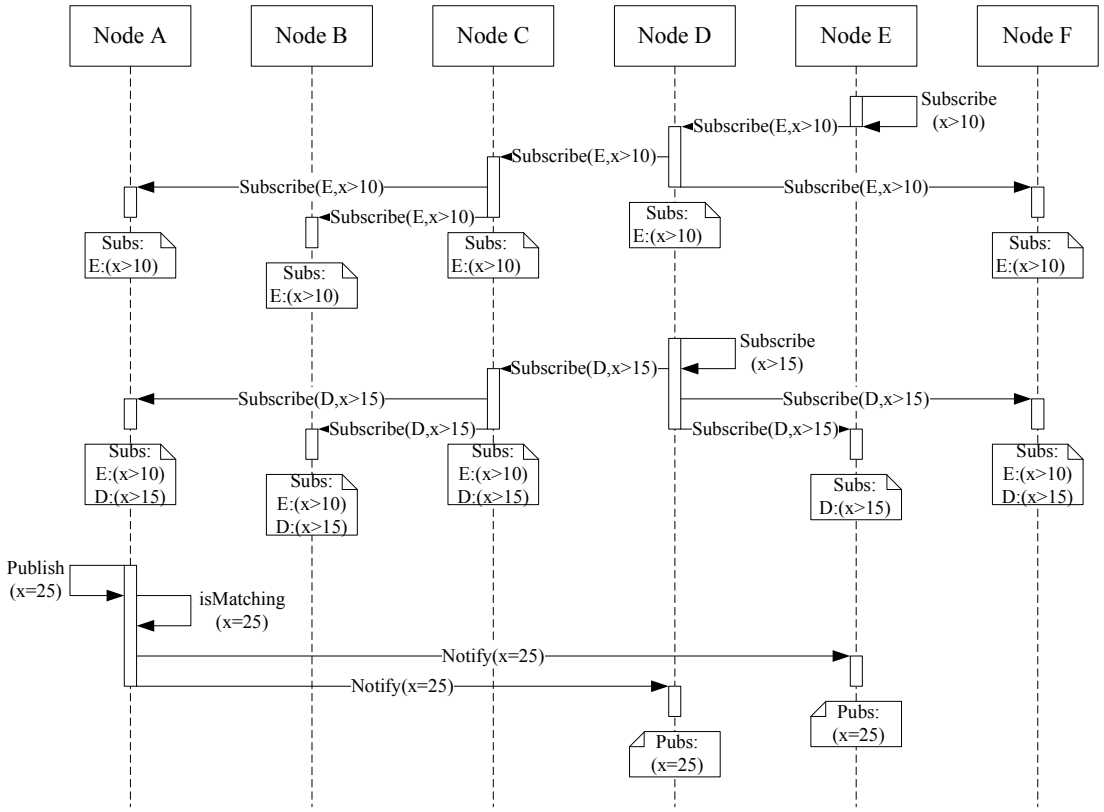



Figure 6.4: Sequence of events in a distributed Boolean system with subscription flooding.

subscriptions of nodes E and D are not propagated through the overlay network since they are stored at their subscriber nodes. Node A publishes a publication $\{x = 20\}$ by flooding the overlay network with it. As we can see, the process of matching begins at each node once the publication reaches it, and is exclusively intended for its local subscriptions. The propagation of this publications stops when it reaches all nodes in this system. In this example, the publication of node A is matching for the subscriptions of nodes E and D.

### 6.2.2 Subscription Flooding

In a distributed Boolean system with subscription flooding, the overlay network is flooded with every new subscription update (i.e. activation or cancellation). As each node stores all active subscriptions in the system, we can use subscription covering and (perfect) merging at the subscriber side to reduce the number of forwarded (i.e. stored) subscriptions in the system. In this routing strategy, publishers perform the matching between publications they publish and all active subscriptions in the system. Besides that, they are also responsible to directly forwarding (i.e. using the network layer protocol) matching publications to their subscribers. This strategy is best suited to environments that should share the processing load (due the matching) in situations where the majority of nodes are also publishers. However, if the frequency of subscription updates is too large, the overlay network will be overloaded with such updates.

**Example 6.2** (Distributed Boolean publish/subscribe system with subscription flooding)**.** Figure 6.4 shows an example sequence of events in the distributed Boolean system with subscription flooding whose topology is shown in Figure 6.2. This sequence is analogous to the previous example and to the examples in Chapter 5 because node E first subscribes to publications that satisfy the condition $\{x > 10\}$, then node D subscribes to publications for which $\{x > 15\}$, and finally, node A publishes a publication $\{x = 20\}$. We can see that the overlay network is completely flooded with subscriptions of nodes E and D. Node A publishes $\{x = 20\}$, sequentiality matches it to the active subscriptions in the system, and then notifies clients E and D whose subscriptions are satisfied with this publication.

The filtering of uninteresting publications at the publisher side is an important characteristic and a big advantage of this routing strategy. However, similarly to the case of centralized Boolean systems with subscription flooding, three problems arise when this routing strategy is used: 1) a publisher and subscriber are actively participating in the interaction at the same time 2) subscribers and publishers do not remain anonymous with respect to each other and 3) the matching happens at the publisher side which is not under control of subscribers and thus can be easily abused.

### 6.2.3 Covering-based Routing

Covering-based routing in distributed Boolean systems is very similar to subscription flooding. A subscription is propagated through an overlay network only if it is not *covered*[3] by a previously propagated and still active subscription. Otherwise, the overlay network will be flooded with the subscription. Every node in the overlay network knows identifiers of its neighbors and all non-covered subscriptions which it has received from them. A newly published publication is processed, for a subscription, at every node on the reverse path from the subscriber to the publisher. In other words, when a new publication is published, the publisher node forwards this publication to those of its neighbors from which it has previously received subscriptions that are satisfied with this publication. Then, each of these neighbors repeats this process and forwards the publication to its neighbors with satisfied subscriptions. This process stops when none of the subscriptions stored at a node are satisfied with the publication. When a subscription is canceled, all nodes which store this subscription must check the covering relations between the subscriptions they store. Please note that both perfect and imperfect *merging*[3] of subscriptions can also be used in combination with this routing strategy to reduce the routing information stored at nodes. This routing strategy is best suited to situations when we want to reduce the communication overhead caused by subscription flooding. However, if the interests of subscribers do not overlap, the overlay network will still be overloaded with subscription updates.

**Example 6.3** (Distributed Boolean publish/subscribe system with covering-based routing)**.** Figure 6.5 shows an example sequence of events in the distributed Boolean system with covering-based routing whose topology is shown in Figure 6.2. This sequence is analogous to the previous examples because node E first subscribes to publications satisfying the condition $\{x > 10\}$, then node D subscribes to publications for which $\{x > 15\}$, and finally, node A publishes a publication which has $\{x = 20\}$.

[3]The covering and merging of subscriptions is explained in Chapter 4.

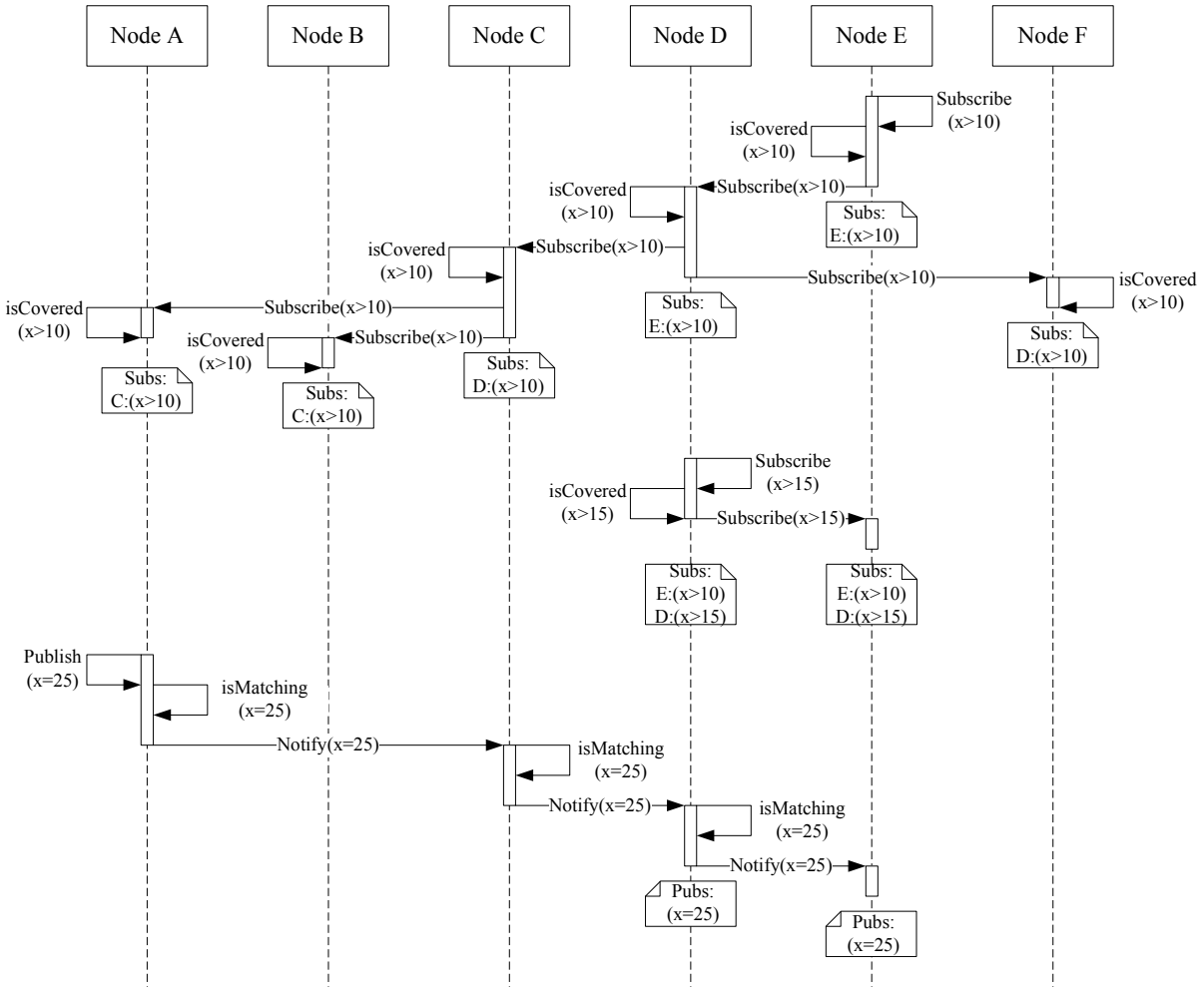



Figure 6.5: Sequence of events in a distributed Boolean system with covering-based routing.

We can see that the overlay network is completely flooded with the subscription of node E, but is not with the subscription of node D since the latter subscription is covered with the former. Please note that this routing strategy tracks neighbors from which propagating subscriptions are received. For example, although node E has already subscribed to node D with subscription $\{x > 10\}$, node D also must subscribe to node E with subscription $\{x > 15\}$. When node A publishes a publication $\{x = 20\}$, every node on the reverse path from the subscribers of this publication to node A, performs the matching between this publication and subscriptions it stores. The propagation of this publication stops when all its subscribers are notified, i.e. when node D delivers this publication to node E.

### 6.2.4 Rendezvous Routing

In every distributed publish/subscribe system with rendezvous routing, the following two methods exists: 1) a method which maps each subscription to an overlay node and 2) a method which maps each publication to an overlay node. A node to which a publication or subscription is mapped to is called *rendezvous*

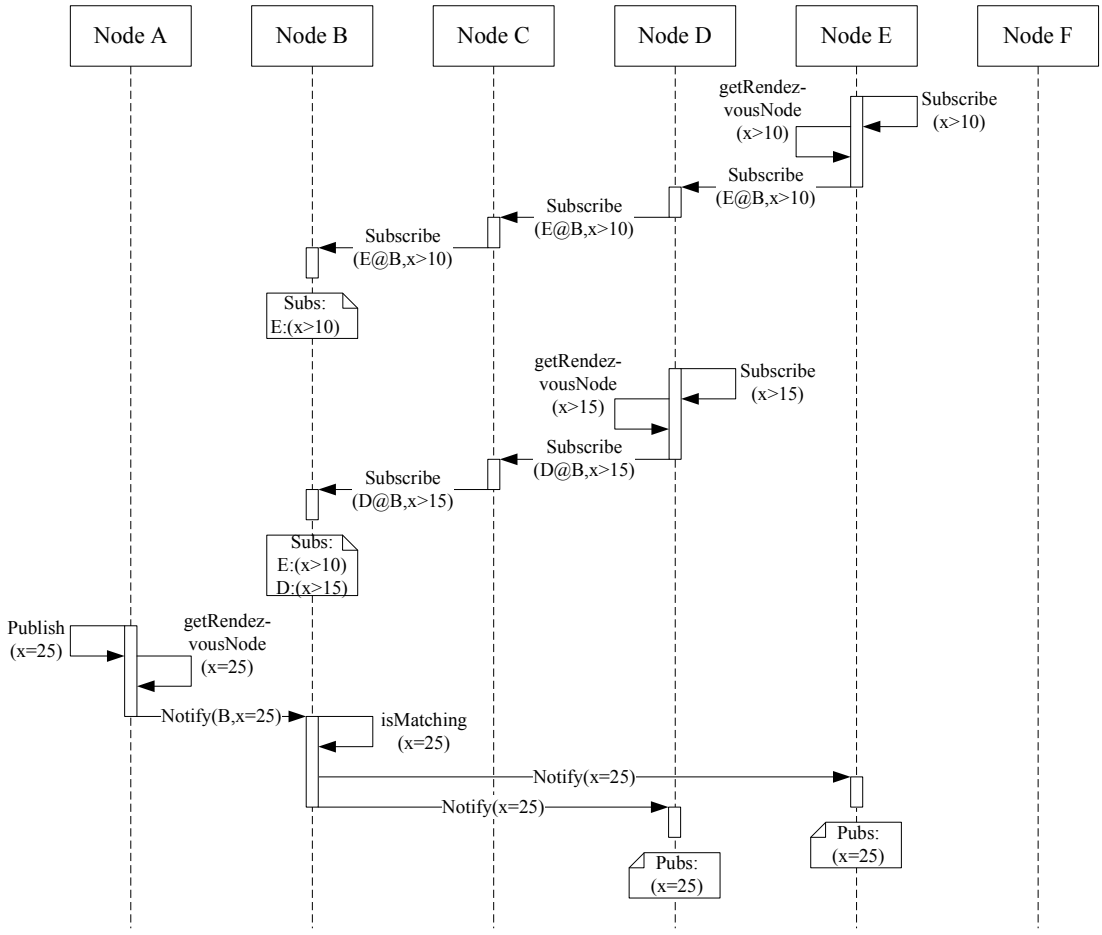



Figure 6.6: Sequence of events in a distributed Boolean system with rendezvous routing.

node. When a new subscription is activated, the subscriber node first determines the subscription rendezvous node, and then through the overlay network it forwards to the rendezvous node its own identifier together with this subscription. Similarly, when a publication is published, its publisher first determines the publication rendezvous node, and then through the overlay network it forwards this publication to the rendezvous node. Upon receiving an incoming publication, a rendezvous node performs the matching between this publication and stored subscriptions to see if this publication matches any of them. Besides that, this node is also responsible to directly deliver (i.e. using a network layer protocol) matching publications to their subscribers. This strategy is best suited to situations when we want to balance processing and memory loads among nodes due to the matching and subscription storing, respectively. Additionally, when this routing strategy is used, no redundant matching needs be performed in the system, which is a big advantage when compared to the covering-based routing. However, the main drawback of this routing strategy is the restricted expressiveness of compatible subscription languages.

**Example 6.4** (Distributed Boolean publish/subscribe system with rendezvous routing)**.** Figure 6.6 shows an example sequence of events in the distributed Boolean system with rendezvous routing whose topology is shown in Figure 6.2. This sequence is analogous to the previous examples because node E first sub-

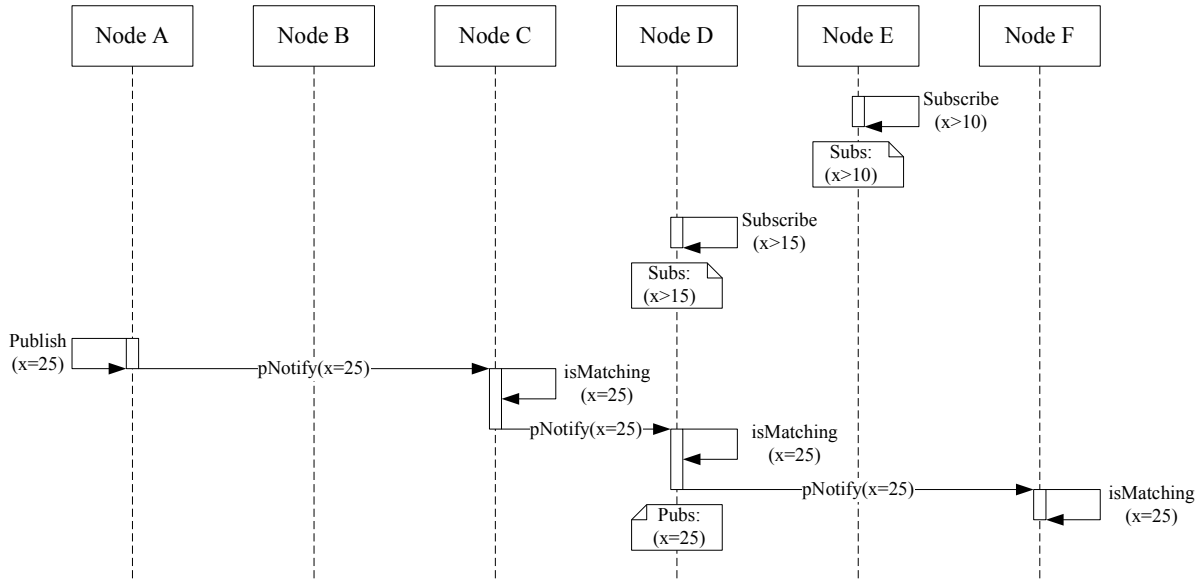



Figure 6.7: Sequence of events in a distributed Boolean system with basic gossiping.

scribes to publications which have $\{x > 10\}$, then node D subscribes to publications satisfying condition $\{x > 15\}$, and finally, node A publishes a publication which has $\{x = 20\}$. We can see that subscriptions of nodes E and D are sent through the overlay network to node B, which is the rendezvous node of these subscriptions. When node A publishes a publication $\{x = 20\}$, this publication is sent through the overlay network to node B, which is its rendezvous node. Upon receiving publication $\{x = 20\}$, node B performs the matching between this publication and stored subscriptions. As this publication is matching for subscriptions of nodes E and D, node B directly notifies these nodes about the published publication.

### 6.2.5 Basic Gossiping

Basic gossiping, as a routing strategy in distributed Boolean publish/subscribe systems, is similar to publication flooding since subscriptions are stored at subscriber nodes and are not propagated further into the overlay network. When a publication is published, it spreads through an overlay network as a gossip (i.e. epidemic). More precisely, in each round of publication spreading, one or a few nodes that have received this publication previously, spread it further to some of their neighbors chosen at random. Upon receiving a newly published publication, a node performs the matching between this publication and its own subscriptions to see if this publication matches any of them, and after that randomly choses one or a few of its neighbors to which it will spread (i.e. forward) this publication. Publication spreading through an overlay network stops after a predefined number of rounds. This routing algorithm is probabilistic (i.e. randomized) and thus does not guarantee publication delivery to all interested subscribers.

**Example 6.5** (Distributed Boolean publish/subscribe system with basic gossiping)**.** Figure 6.7 shows an example sequence of events in the distributed Boolean system with basic gossiping whose topology is shown in Figure 6.2. This sequence is analogous to the previous examples because node E first subscribes to publications satisfying the condition $\{x > 10\}$, then node D subscribes to publications for which

the following holds $\{x > 15\}$, and finally, node A publishes a publication $\{x = 20\}$. We see that subscriptions of nodes E and D are not propagated into the overlay network, because they are stored at their subscriber nodes. When node A publishes a publication $\{x = 20\}$, this publication randomly spreads (*pNotify*) through the overlay network. In the first round it reaches node C, in the second round node E, and finally in the third round node F. Therefore, the spreading of this publication stops after 3 rounds. As previously explained, the process of matching begins when a node receives this publication, and is exclusively intended for its local subscriptions. We see that node D is notified about the publication of node A, but node E is not, although this publication matches its subscription.

When selective routing strategies (i.e. covering-based and rendezvous routing) are used, subscriptions are not stored at the publisher or subscriber side, but at the third side. Therefore, in distributed Boolean systems with a high churn rate, these two strategies cannot guarantee the delivery of matching publications due to the reconfiguration of the overlay network. For this reason, probabilistic gossiping is used in such systems to improve the reliability of publication delivery [182].

### 6.2.6 Informed Gossiping

Informed gossiping, as a routing strategy in distributed Boolean systems, is very similar to basic gossiping since both strategies are probabilistic. For both routing strategies, nodes store their own subscriptions, but in the case of informed gossiping each node additionally stores subscriptions of its close neighbors. In this way, besides performing the matching of an incoming publication for its own subscriptions, each node also performs the matching for the subscriptions of its neighbors. This is the deterministic part of publication spreading. Additionally, after finishing the deterministic publication spreading, the node can also start the probabilistic part of spreading by randomly choosing one or a few of its neighbors to which it will forward this publication. Similar to basic gossiping, the spreading of a publication through the overlay network stops after a selected number of probabilistic rounds. Therefore, while basic gossiping is purely probabilistic, informed gossiping is partially probabilistic and partially deterministic. Due to the deterministic part, the probability to successfully deliver a publication to subscribers is increased in the case of informed gossiping when compared to basic gossiping. Again, this strategy is best used in systems with a high churn rate to improve the reliability of publication delivery.

**Example 6.6** (Distributed Boolean publish/subscribe system with informed gossiping)**.** Figure 6.8 shows an example sequence of events in the distributed Boolean system with informed gossiping whose topology is shown in Figure 6.2. This sequence is analogous to the previous examples because node E first subscribes to publications which have $\{x > 10\}$, then node D subscribes to publications for which $\{x > 15\}$, and finally, node A publishes a publication which has $\{x = 20\}$. We can see that subscriptions of nodes E and D propagate just one hop away from these nodes. When node A publishes a publication $\{x = 20\}$, this publication spreads both randomly (*pNotify*) and deterministically (*dNotify*) through the overlay network. In the first round it probabilistically reaches node C, which has the subscription of node D stored in memory. Therefore, in the second round node C deterministically forwards this publication to node

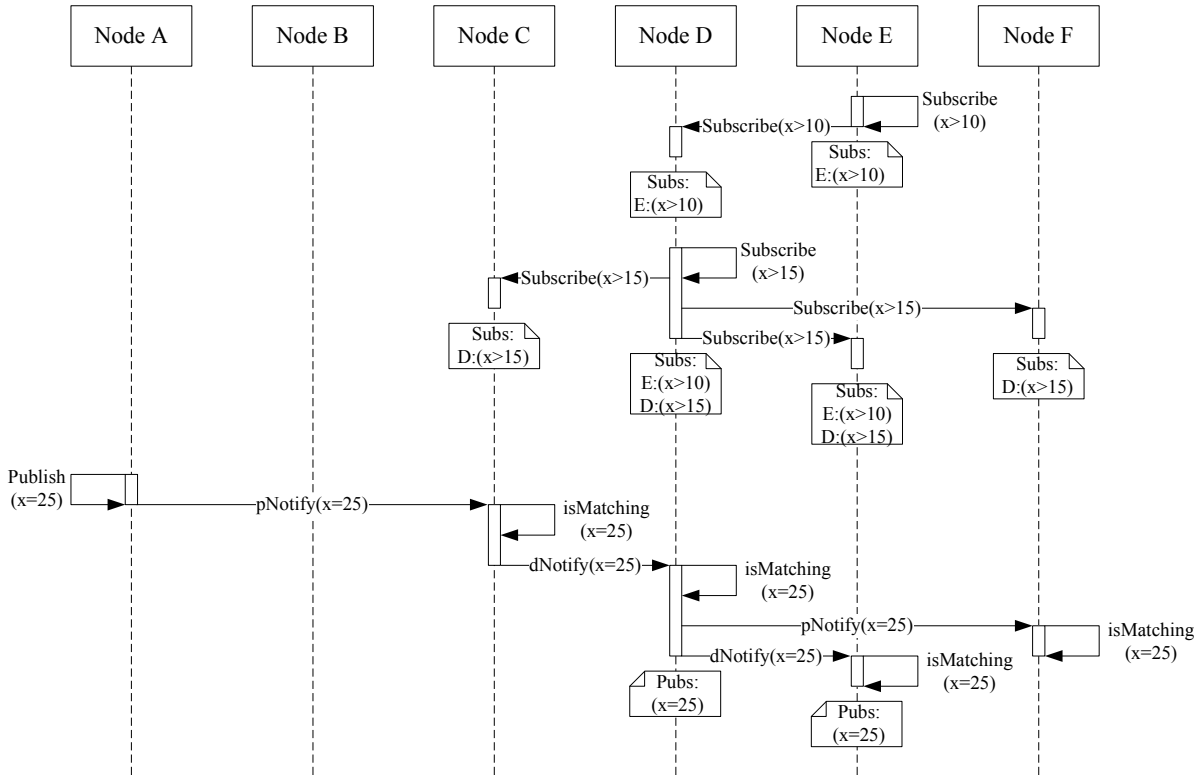



Figure 6.8: Sequence of events in a distributed Boolean system with informed gossiping.

D. In the third round, the publication probabilistically spreads to node F, and deterministically to node E. Identically as in the basic gossiping example, the spreading of this publication also stops after 3 rounds.

## 6.3 Routing Strategies in Distributed Top-k/w Publish/Subscribe Systems

In the previous section we have presented six routing strategies for distributed Boolean publish/subscribe systems. In this section we adapt these routing strategies to distributed top-k/w publish/subscribe systems (distributed top-k/w systems). For each of the strategies, we explain differences when compared to the original routing strategy and identify locations of top-k/w processors and recent buffers in the system. As we will see, two routing strategies introduce significant additional communication overhead in distributed top-k/w systems when compared to distributed Boolean systems, while the other four strategies do not introduce such overhead.

From Chapter 4 we know that every top-k/w proxy subscription consists of a static part and a threshold, which is the dynamic part. In practice, the threshold of a top-k/w subscription always fluctuates in time around a central value. It is very important to notice that a more critical situation is when a subscription subspace of interest expands than the opposite situation since in the former situation the subscriber could miss some matching publications if this information is not instantly propagated through the overlay network. In the latter situation the subscriber could be notified with non-matching publications, but as in practice such publications can be filtered at the subscriber side, this is a less problematic situation.

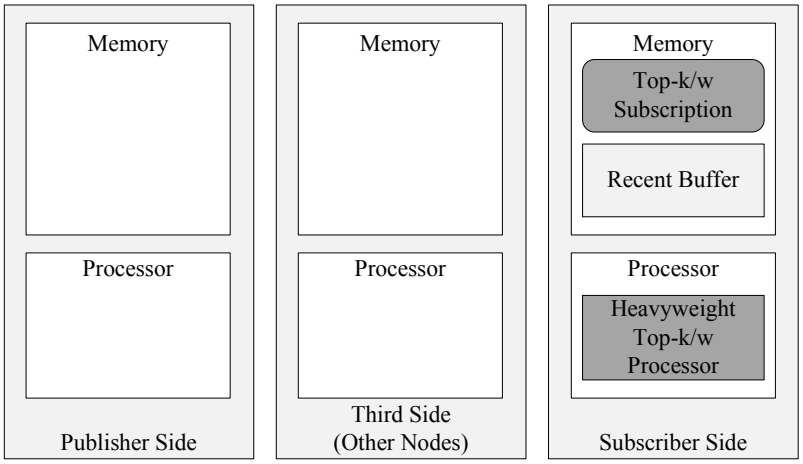



Figure 6.9: Distributed top-k/w publish/subscribe system with publication flooding (and basic gossiping).

### 6.3.1 Publication Flooding

Figure 6.9 shows entities and their roles in a distributed top-k/w system with publication flooding. As we can see, each subscriber has a heavyweight top-k/w processor and recent buffer to process the stream of incoming publications for its own subscriptions. This processor knows current thresholds of subscriptions it stores in memory and thus can without any doubt detect matching publications for these subscriptions. In this routing strategy we do not need to store proxy subscriptions in the network since the matching is performed at the subscriber side. We conclude that this routing strategy is well suited for distributed top-k/w systems because it does not introduce any additional communication overhead when compared to distributed Boolean systems.

### 6.3.2 Subscription Flooding

Figure 6.10 shows entities and their roles in a distributed top-k/w system with subscription flooding. As we can see, each publisher node stores proxy subscriptions of all active top-k/w subscriptions in the system and thus must have a lightweight top-k/w processor and recent buffer to process the stream of its own publications for these proxy subscriptions. The matching between a published publication and stored proxy top-k/w subscriptions is performed twice at the publisher side[4]. The first matching is performed immediately after the publishing, whereas the second is performed immediately after dropping this publication from the recent buffer. After the first matching, the publisher directly (i.e. using a network layer protocol) forwards the publication to subscribers with satisfied subscriptions. However, after the second matching it directly forwards this publications only to those subscribers whose subscriptions are now satisfied, but were not during the first matching. Each subscriber must have a heavyweight top-k/w processor and recent buffer to process the stream of received publications for its own subscriptions. When a subscription threshold changes, all nodes in the overlay network have to be informed about this change to update their proxy subscriptions. We conclude that this routing strategy can be implemented for distributed top-k/w systems, but is not well suited for them since the synchronization of subscription

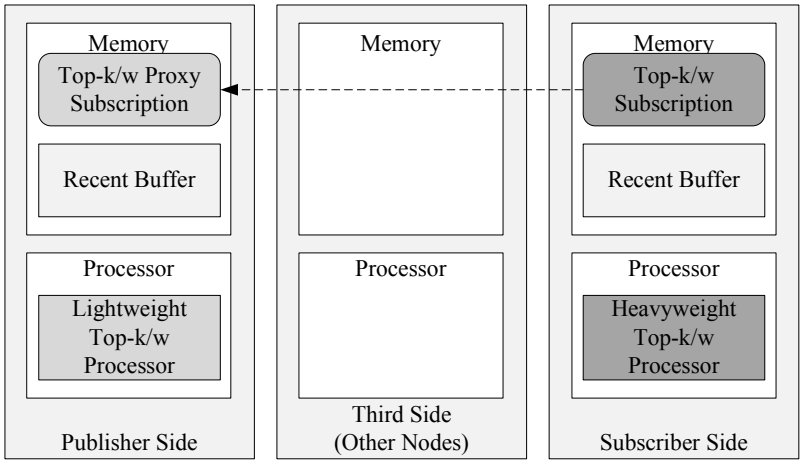



Figure 6.10: Distributed top-k/w publish/subscribe system with subscription flooding.

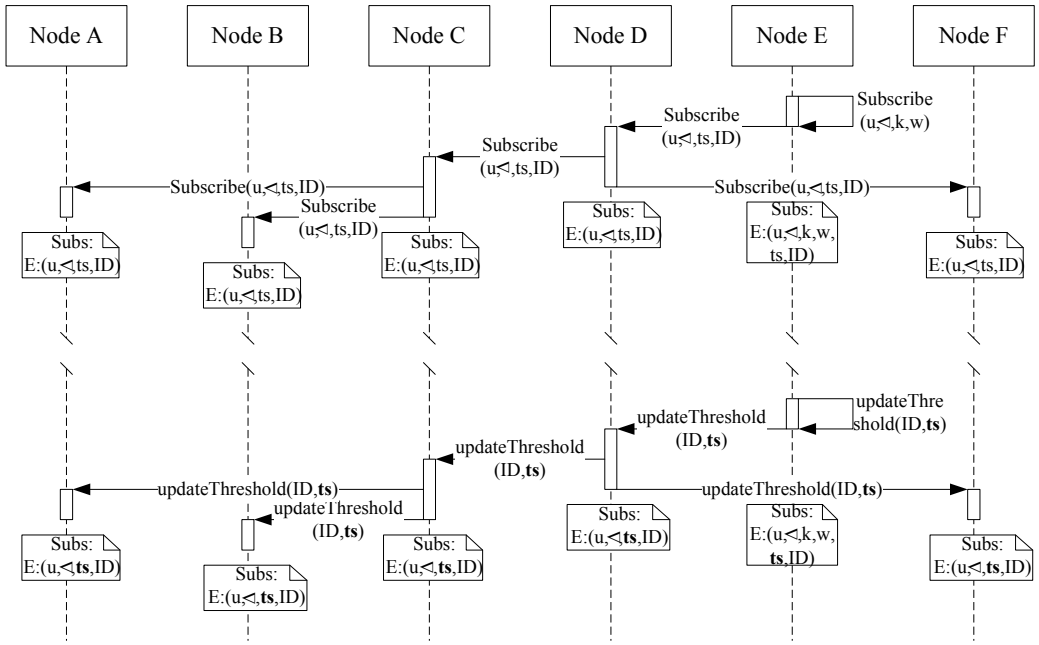



Figure 6.11: Sequence of events in a distributed top-k/w system with subscription flooding.

thresholds introduces a large communication overhead when compared to distributed Boolean systems.

**Example 6.7** (Distributed top-k/w publish/subscribe system with subscription flooding)**.** Figure 6.11 shows a sequence of events in the distributed top-k/w system with subscription flooding whose topology is shown in Figure 6.2. This example shows how information about a threshold change spreads through this system. In this example, node E first subscribes by flooding the overlay network with the proxy subscriptions of its top-k/w subscription. After some time, when the threshold of this subscription changes,

[4]These two matchings are analogous the two insertion attempts for SA and RA with filters, which are explained in Chapter 3. In the case of PA, we do not need a recent buffer at the publisher side since the matching is performed only once, immediately after the publishing.

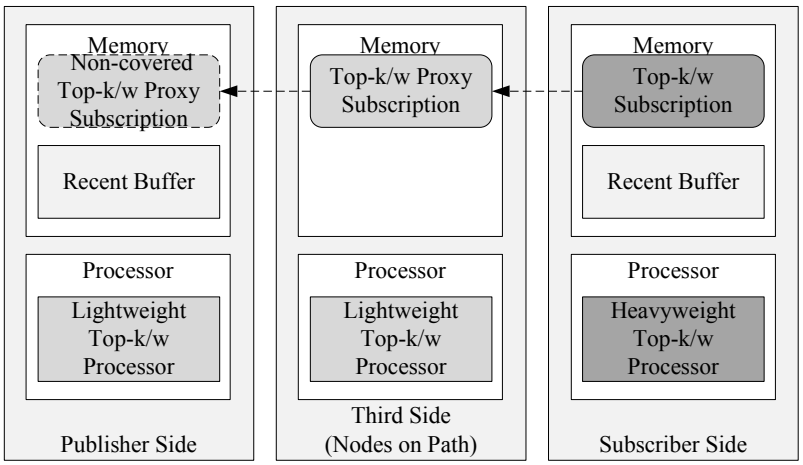



Figure 6.12: Distributed top-k/w publish/subscribe system with covering-based routing.

node E again has to flood the overlay network to inform all other nodes in the system to update their proxy subscription.

### 6.3.3 Covering-based Routing

When a new subscription is activated in a distributed top-k/w system with covering-based routing, the overlay network is flooded with the corresponding proxy subscriptions. However, if such a proxy subscription is covered in a direction with one or more of the previously propagated proxy subscriptions, its propagation will be blocked in that direction. Therefore, as shown in Figure 6.12, each node on the reverse path from a subscriber to any publisher must have a lightweight top-k/w processor to process the stream of incoming publications for proxy subscriptions[5] of the subscriber. Additionally, each subscriber must have a heavyweight top-k/w processor to process the stream of received publications for its own subscriptions, and each publisher must have a lightweight top-k/w processor and recent buffer to process the stream of its own publications for proxy subscriptions that it has received previously. As in the case of subscription flooding, the matching between a published publication and stored proxy top-k/w subscriptions is performed twice at the publisher side. The first matching is performed immediately after the publishing, whereas the second is performed immediately after the dropping of this publication from the recent buffer. After both matchings, the publisher forwards the publication to those of its neighbors from which it has received satisfied top-k/w proxy subscriptions. This is necessary, because new subscribers could appear between these two matchings.

When either a subscription is canceled or its threshold changes, all nodes which store its proxy subscription should check covering relations between all proxy subscriptions they store. However, this checking is not trivial and may produce large processing overhead at nodes in an overlay network. In other words, not only does this routing strategy flood the overlay network with frequent threshold changes, but it additionally requires rechecking of subscription coverings after each such change. For this reason we

[5] If a proxy subscription is covered, the node will process the proxy subscription that covers it

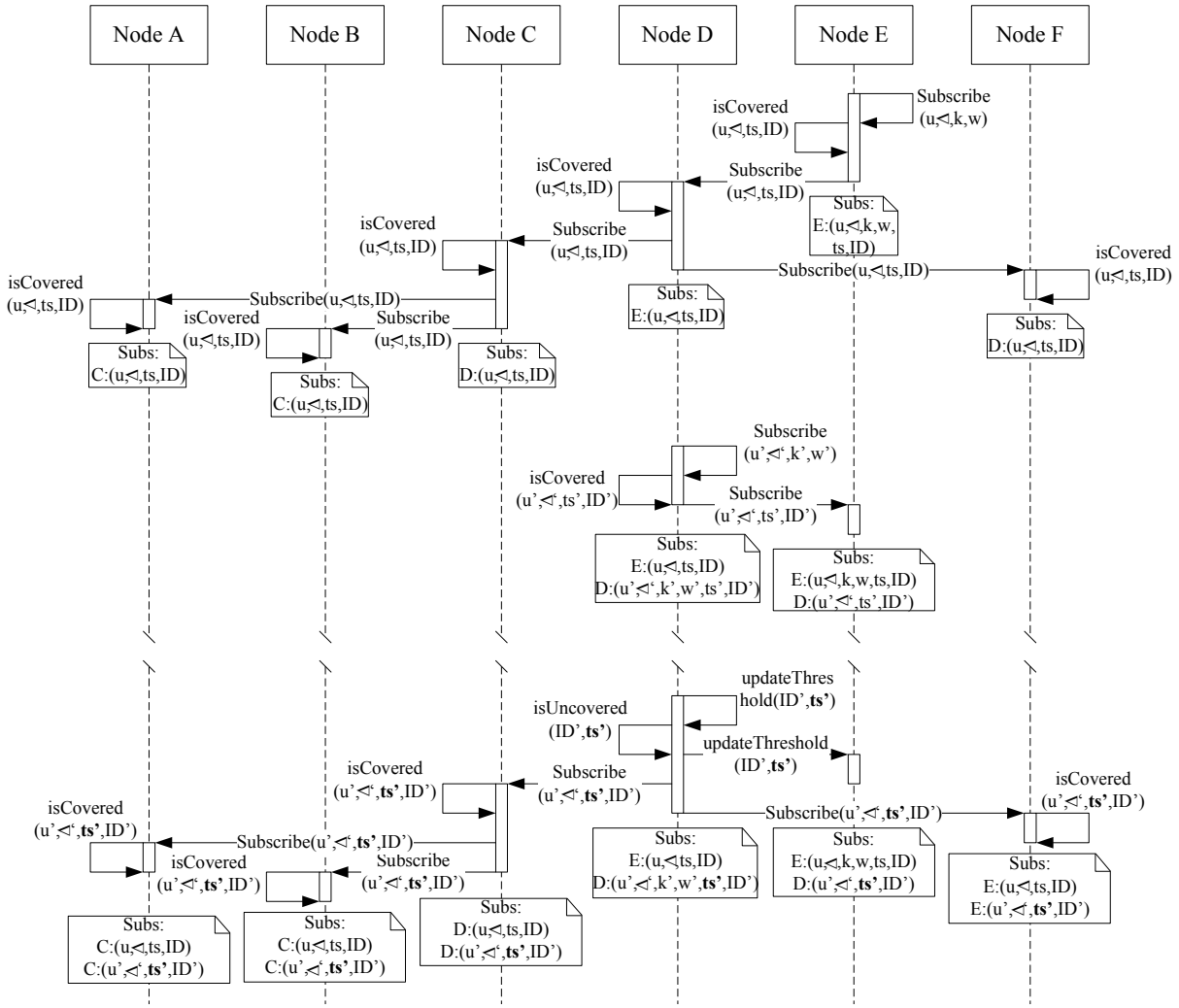



Figure 6.13: Sequence of events in a distributed top-k/w system with covering-based routing.

conclude that this routing strategy can be implemented for distributed top-k/w systems, but is not well suited for them since the synchronization of subscription thresholds introduces a large processing and communication overhead when compared to distributed Boolean systems. Please note that subscription merging can be used with this routing strategy to reduce the routing information stored at nodes, but it would introduce an additional processing overhead in a distributed top-k/w system.

**Example 6.8** (Distributed top-k/w publish/subscribe system with covering-based routing)**.** Figure 6.13 shows a sequence of events in the distributed top-k/w system with covering-based routing whose topology is shown in Figure 6.2. This example shows how information about a threshold change spreads through this system. As we can see, two nodes activate new subscriptions, first node E, and then node D. The overlay network is completely flooded with proxies of the subscription of node E because this subscription is not covered by any other. On the contrary, node D forwards the proxy of its subscription just to node E because this subscription is covered with the proxy subscription of node E. After some time, the subspace

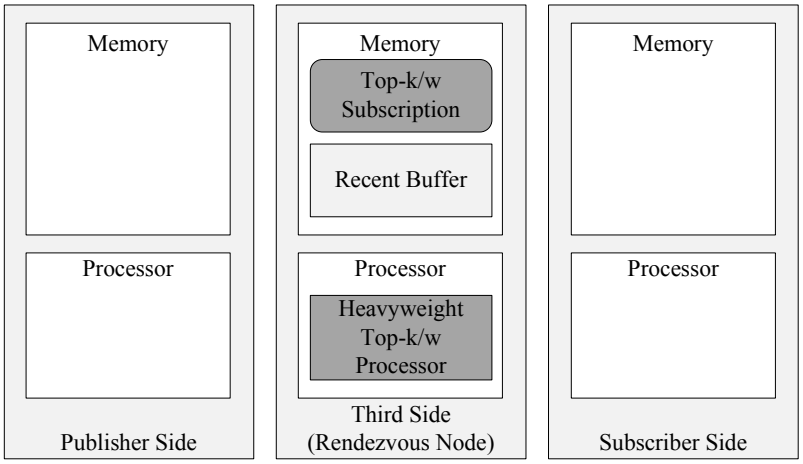



Figure 6.14: Distributed top-k/w publish/subscribe system with rendezvous routing.

of the subscription of node D expands and other nodes in the system have to be informed about this change. The part of the overlay network that already stores the corresponding proxy subscription (i.e. node E) receives just the new threshold value, while the other part (i.e. nodes C and D, and then nodes A and B) receives the whole proxy subscription.

### 6.3.4 Rendezvous Routing

Figure 6.14 shows entities and their roles in a distributed top-k/w system with rendezvous routing. As we can see, each rendezvous node in the overlay network must have a heavyweight top-k/w processor and recent buffer. Such a processor preforms the matching between incoming publications mapped to its node and subscriptions it stores (i.e. which were also mapped to its node). This processor knows current thresholds of subscriptions it stores in memory, and thus can without any doubt detect matching publications for these subscriptions. In this routing strategy we do not need to store proxy subscriptions in the network since subscriptions are stored only at their rendezvous nodes. We conclude that this routing strategy is well suited for distributed top-k/w systems because it does not introduce[6]any additional communication overhead when compared to distributed Boolean systems.

### 6.3.5 Basic Gossiping

Figure 6.9 shows entities and their roles in a distributed top-k/w system with basic gossiping. Similar to the case of publication flooding, each subscriber node must have a heavyweight top-k/w processor and recent buffer to process the stream of incoming publications for its local subscriptions. This processor knows current thresholds of subscriptions it stores in memory and thus can without any doubt detect matching publications for these subscriptions. In this routing strategy we do not need to store proxy subscriptions in the network since the matching is performed at the subscriber side. We conclude that

[6]However, if rendezvous nodes have to contact other nodes in the overlay network due to the processing of publications, the additional communication overhead can be generated in top-k/w systems. For example, this happens in D-ZaLaPS, a distributed top-k/w system based on rendezvous routing, which we present later in this chapter.

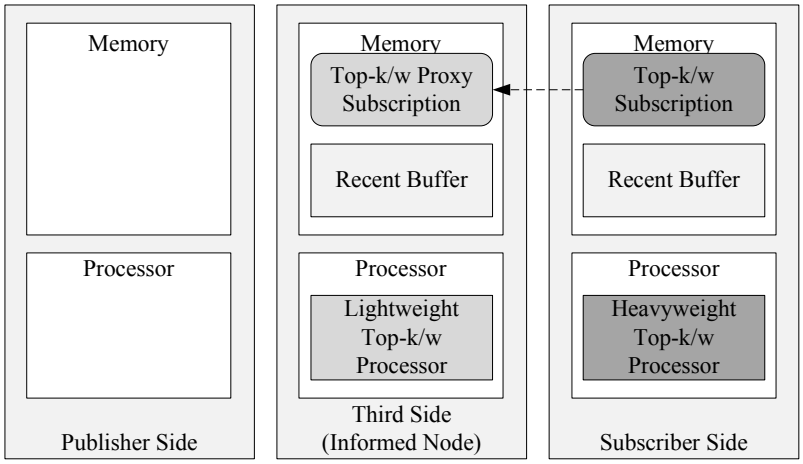



Figure 6.15: Distributed top-k/w publish/subscribe system with informed gossiping.

this routing strategy is well suited for distributed top-k/w systems because it does not introduce any additional communication overhead when compared to centralized Boolean systems.

### 6.3.6 Informed Gossiping

Figure 6.15 shows entities and their roles in a distributed top-k/w system with informed gossiping. Similarly to the case with basic gossiping, each subscriber must have a heavyweight top-k/w processor and recent buffer to process the stream of incoming publications for its local subscriptions. A key characteristic of this strategy is that each subscriber node forwards proxies of its own subscriptions to its close neighbors. Therefore, each close neighbor must have a lightweight top-k/w processor and recent buffer to process the stream of incoming publications for proxy subscriptions of its close neighbors. Similarly as in the case of subscription flooding and covering-based routing, a node which receives a publication during its spreading through the overlay network stores it in the recent buffer and performs two matchings. The first matching is performed immediately after receiving this publication, whereas the second is performed immediately after dropping this publication from the recent buffer. After the first matching, the node directly forwards the publication to its close neighbors with satisfied subscriptions. However, after the second matching it directly forwards this publications only to those of its close neighbors whose subscriptions are now satisfied, but were not during the first matching. These proxy subscriptions stored at neighboring nodes have to be synchronized with their originals, but since the information about a threshold change is forwarded just locally, we conclude that this routing strategy is well suited for distributed top-k/w systems. However, when compared to the case of a distributed Boolean system, an additional message overhead exists due the local synchronization of subscription thresholds between neighboring nodes.

**Example 6.9** (Distributed top-k/w publish/subscribe system with informed gossiping)**.** Figure 6.16 shows a sequence of events in the distributed top-k/w system with informed gossiping whose topology is shown in Figure 6.2. This example shows how information about a threshold change spreads through this system.

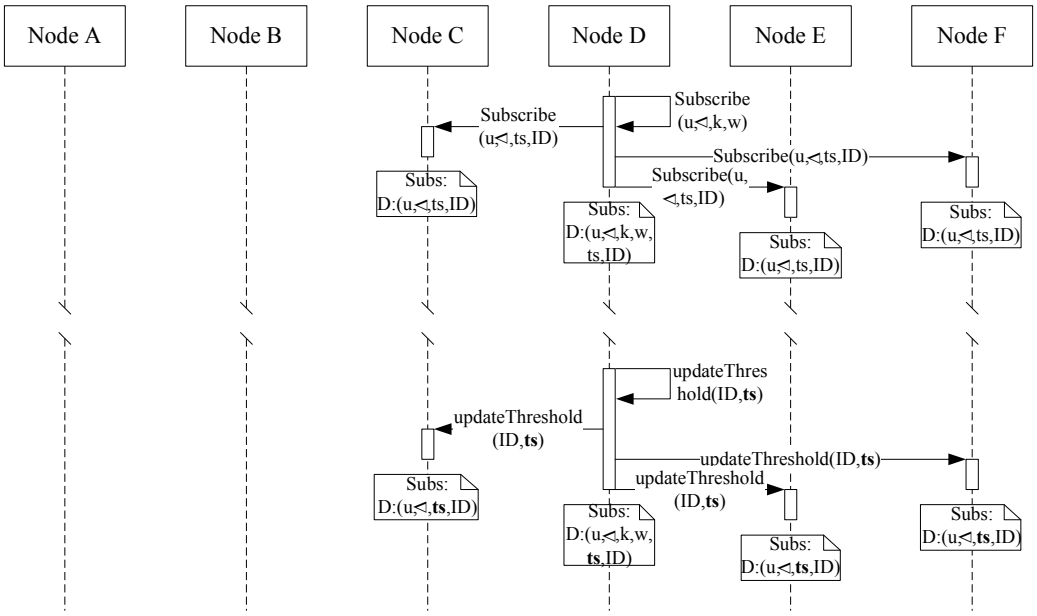



Figure 6.16: Sequence of events in a distributed top-k/w system with informed gossiping.

As we can see, node D subscribes by forwarding the proxies of its subscription to neighboring nodes (one hop away). After some time, when the threshold of this subscription changes, node D informs its close neighbors to update the corresponding proxies.

## 6.4 D-ZaLaPS: A Distributed Top-k/w Pub/Sub System Supporting Distance Scoring Functions

In this section we present D-ZaLaPS[7], a distributed rendezvous-based top-k/w publish/subscribe system supporting distance scoring functions which is built on top of the CAN structured peer-to-peer overlay [178]. D-ZaLaPS supports (ranking) Boolean and top-k/w subscriptions with distance scoring functions.

We assume that both subscriptions and publications are represented as points in a $d$-dimensional Euclidean space. The score of a publication $p$ with respect to subscription $s$ is calculated using the following formula: $u_s(p) = d(p_s, p_p) = [\sum_{i=1}^{d}(v_i - v_i)^2]^{\frac{1}{2}}$, where $p_p = \{v_1, v_2, \ldots, v_d\}$ and $p_s = \{v_1, v_2, \ldots, v_d\}$ are points representing $p$ and $s$, respectively. The score comparator $\overset{s}{\rhd}$ is defined such that lower scores imply higher ranks.

The CAN divides a $d$-dimensional Euclidean space, which is located on a $d$-torus, between peers in a peer-to-peer overlay network by assigning non-overlapping zones of this space to peers. It also ensures that each peer is connected to the peers in neighboring zones, and that each part of the Euclidean space has a responsible peer. Every peer maintains a routing table that stores the information about the IP address and assigned zone for each of its neighbors. This purely local information is sufficient to route a message between any two arbitrary points in this space: Each peer on the path from the source to destination peer,

[7] Zagreb-Lausanne distributed top-k/w Publish/Subscribe system supporting Distance scoring functions.

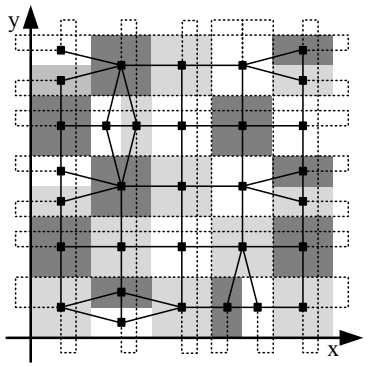



Figure 6.17: An example Content-Addressable Network (CAN).

forwards a message to a selected neighbor that is, in the Euclidean space, closest to the destination point. CAN also supports dynamic joining/leaving of peers, and enables load balancing such that the Euclidean space is divided according to the true load on the peers, and not evenly.

**Example 6.10.** An example CAN network consisting of 32 peers that divide a 2-dimensional Euclidean space is shown in Figure 6.17. The zone of each peer is depicted in a different color compared to the zones of its neighbors, while each peer is represented by a black dot. Note that peer zones have different sizes, and that every peer is connected to its neighbors, which is depicted to solid and dotted lines. Dotted lines connect cross-border neighboring peers because the Euclidean space is located on a 2-dimensional torus where the space wraps around its borders. Such network forms the overlay network of our distributed processing system, and each peer represents a centralized top-k/w processor.

Without the loss of generality[8], D-ZaLaPS additionally partitions Euclidean space to equally sized cells (as in the regular grid) such that each peer is responsible for those cells whose center points belong to its zone of responsibility. In this way, D-ZaLaPS uses CAN to route messages from one cell to any other cell in the Euclidean space. This additional partitioning is necessary to reduce the message overhead when updating proxy subscription thresholds, which is going to be explained later in this section.

Let us now discuss how D-ZaLaPS divides a $d$-dimensional Euclidean space between the peers, and assigns responsibility for subscription activation/cancellation and data processing. It partitions the Euclidean space to cells of equal size as in the regular grid to identify cells of interest for a subscription, and assigns responsibility to peers over zones comprising a set of cells. Both subscriptions and publications are also assigned to a cell from the Euclidean space based on the point which represents it, and each peer is responsible for those subscriptions and publications assigned to cells that fall within its zone of responsibility. In this way, D-ZaLaPS uses CAN to route messages from one cell to any other cell in the Euclidean space. The additional partitioning of the Euclidean space is necessary to reduce message overhead when updating subscription thresholds, as explained later in this section. Our approach can be further generalized to support zones comprising cells of different sizes: Each peer would have to

[8]To generalize our approach each peer has to partition its zone to cells and inform its neighbors about the way it performed the partitioning of its zone.

partition its zone of responsibility and then inform its neighbors about the partitioning of its zone. Such partitioning can prevent cell overload, while CAN solves the problem of load balancing at the peer level. However, to simplify the presentation, let us assume that cells are of equal size and that they correspond to cells within the regular grid.

**Example 6.11.** For example, in Figure 6.17 we partitioned the Euclidean space to 100 equally sized cells such that each peer is responsible for either 2 or 4 cells.

### 6.4.1 Routing Messages in D-ZaLaPS

Let us now explain the routing algorithm behind D-ZaLaPS. Each peer is the *rendezvous peer* for all subscriptions and publications belonging to its cells of responsibility. In this way, each newly published publication and newly activated subscription will be routed to the corresponding rendezvous peer. A rendezvous peer stores received subscriptions and processes incoming publications for them. Any peer may publish a publication and such a peer is the *publisher peer* for the published publication. Analogously, any peer can also activate a subscription and then it will be the *subscriber peer* for the activated subscription.

Note that a subscription which is under the responsibility of a peer may cover the cells of interest outside the zone of this peer responsibility, and recall that the cells of interest of a top-k/w subscription usually change over time. Moreover, since each rendezvous peer is responsible for one or more subscriptions, such situation is even more probable. Therefore, a peer will need to interact with its neighboring peers to receive publications of interest for its subscriptions, although the peer is not responsible for zones to which these publications belong to. To reduce the number of exchanged messages between neighboring peers, each peer *merges* the subscriptions it is responsible for in a special *merger* which covers all subspaces of interest of the merged subscriptions. The subspace of interest of such a merger is always converted to cells, while peers which are responsible for these cells are additionally informed with a *merger update message* that a rendezvous peer is interested in publications belonging to their cells. Therefore, such peers have to forward publications to interested rendezvous peers for further processing. Without the loss of generality, and to improve the processing performance at each peer, we assume that mergers are hyper-rectangles of the Euclidean space.

**Example 6.12.** In Figure 6.18 we can see a merger of three top-k/w subscriptions. The shown merger is a hyper-rectangle of an 2-dimensional Euclidean space and covers all the subspaces of interest of the merged subscriptions. It is important to notice that merger cells of interest completely cover the merger.

**Subscription Activation.** When a new subscription is activated, the subscriber peer determines the cell to which its subscription belongs to, and subsequently sends the subscription in a *subscribe message* to the rendezvous peer responsible for this cell. The rendezvous peer then performs merging of the new subscription with other subscriptions under its responsibility. If the newly created merger expands to additional cells due to new subscription activation, the rendezvous peer will inform all peers responsible for the additional cells that it is interested in publications belonging to these cells by sending a *merger update message*.

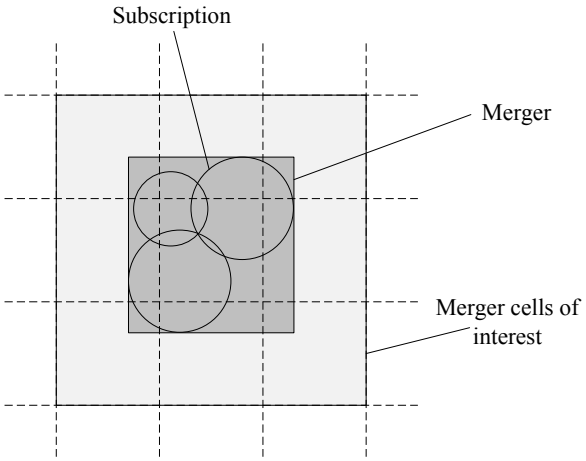



Figure 6.18: The relationship between a subscription, merger and merger cells of interest.

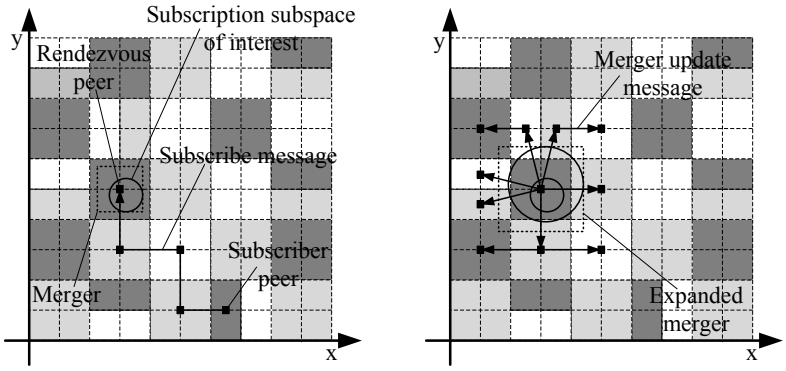
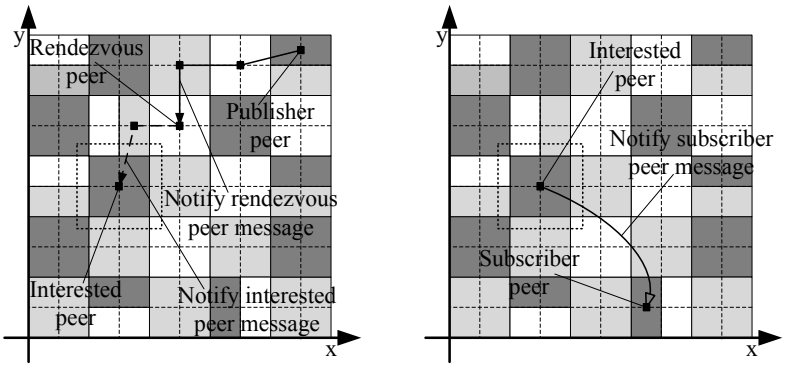



Figure 6.19: A subscription activation in D-ZaLaPS.

**Example 6.13.** Figure 6.19 depicts an example of the top-k/w subscription activation in the CAN network from Figure 6.17. Upon subscription activation, the subscriber peer sends a subscribe message to the responsible rendezvous peer, as shown in Figure 6.19a. In Figure 6.19b we see that the merger of the rendezvous peer has expanded due the subscription activation. The rendezvous peer thus sends merger update messages to the neighboring peers whose cells of responsibility intersect with the merger subspace of interest.

**Subscription Cancellation.** When a subscription is canceled, similarly to subscription activation, the subscriber peer sends an *unsubscribe message* to the rendezvous peer responsible for its previously activated subscription. After receiving this message, the rendezvous peer contracts its merger if necessary, and informs each of the peers that are responsible for cells in which the merger is no more interested with a *merger update message*.

**Merger Expansion and Contraction.** When a merger expands or contracts during publication and subscription processing, the rendezvous peer informs all the affected peers about the change of its interest with a *merger update message*. Analogous to the experimental evaluation in Section 5.3.1, the number




(a) Propagation of notify rendezvous and interested peer messages

(b) Publication forwarding to a subscriber peer

Figure 6.20: A publishing in D-ZalaPS.

of merger contraction and expansion messages would be much larger if the Euclidean space has not been partitioned in cells.

**Publishing.** When a new publication is published, its publisher peer sends it in a *notify rendezvous peer message* to the rendezvous peer responsible for this cell. The rendezvous peer performs top-k/w processing of this publication for active subscriptions its stores to determine if this publication interests any of them. From Section 3.6.1 we know that a top-k/w subscription is interested in a publication if the publication score, with respect to the subscription, is lower than the subscription threshold. When this happens, the rendezvous peer directly forwards the publication to the subscriber peer of such a subscription in a *notify owner peer message* using a network layer protocol. Additionally, it also forwards the publication in a *notify interested peer message* to those peers that have previously expressed interest in the cell to which this publication belongs to. Upon receiving the publication, these peers also perform its top-k/w processing for subscriptions they store, and forward it directly to subscriber peers which have previously issued subscriptions for which it is a top-k/w publication.

**Example 6.14.** Figure 6.20 shows an example of publishing a publication which is a top-k/w for the subscription activated in Example 6.13. As we can see in Figure 6.20a, this publication is first forwarded in a *notify rendezvous peer message* to the corresponding rendezvous peer. Since this peer has previously received a merger update message from an interested peer, it forwards the received publication to the interested peer in a notify interested peer message. The interested peer then processes the received publication and concludes that it is a top-k/w publication for the subscription activated in Example 6.13 and thus forwards it to the subscriber peer in a *notify owner peer message*, as shown in Figure 6.20b.

Please note that in the case when RAPF is used as a processing algorithm on peers, the recent buffer has to be maintained at every rendezvous peer, and is shared by all subscriptions under the peer's responsibility. Analogous to the centralized case, we try to insert a publication into the k-skyband of a top-k/w subscription twice, and thus the rendezvous peer has to notify each of the peers interested in this publication also twice: the first *notify interested peer message* is sent immediately after the publication

Table 6.1: Default values of parameters used in the experimental evaluation of D-ZaLaPS.

| Parameter | Symbol | Value |
|---|---|---|
| Number of publications | $N$ | $10^6$ |
| Number of subscriptions | $m$ | 400 |
| Size of recent buffer | $b$ | 2000 |
| Data dimensionality | $d$ | 4 |
| Grid resolution | $\rho$ | 12 |
| RA: pruning coefficient | $\gamma$ | 0.2 |
| PA: probability of error | $\sigma$ | $10^{-3}$ |
| Number of peers | $C$ | 256 |

arrival to the interested peer, whereas the second is sent after the publication is dropped from the recent buffer.

In the next section we experimentally evaluate performance of D-ZaLaPS in the case of Boolean and top-k/w subscriptions.

## 6.5 Experimental Evaluation of D-ZaLaPS

In this section we present an experimental study comparing the performance and scalability of D-ZaLaPS for Boolean and top-k/w subscriptions. For top-k/w subscriptions we employ two algorithms presented in Chapter 3: PA and RAPF. The former algorithm is probabilistic, while the latter is the fastest of the deterministic top-k/w processing algorithms.

Please note that for the sake of simplicity, we omit subscripts of subscription parameters in this section, if otherwise not explicitly stated.

In the experimental evaluation we use an uniformly generated synthetic dataset of 4-dimensional points within the interval $[0, 1]$. We use these points for both publications and subscriptions. The default scenario used in all experiments is the following: We first connected peers, then for each of the subscriptions we randomly selected a subscriber peer and activated this subscription, and finally we simulated the publishing by randomly selecting a publisher peer for each publication. In other words, we simulated the system with static peers and subscriptions. The default simulation parameters used in experiments are specified in Table 6.1. Differently than in experiments in the previous chapters, we first published $10^5$ publications to let the subscriptions set their thresholds[9], and then published next $N$ publications for which we obtained and analyzed results.

In the following we examine the observed number of exchanged messages and the scalability of D-ZaLaPS for different types of subscriptions.

[9] A top-k/w subscription is subscribed to the whole attribute space when activated, which can generate lots of threshold update messages. However, in a real-life system there are always some publications in recent buffers and in memory and they can be used to set the initial thresholds of newly activated top-k/w subscriptions. For this reason we evaluate the system in a typical situation with already set thresholds.

### 6.5.1 Default Simulation Scenario

In this simulation scenario we compare the number of exchanged messages in D-ZaLaPS, first for PA-based top-k/w subscriptions, then for RAPF-based top-k/w subscriptions, and finally for Boolean subscriptions. Please note that the Boolean threshold value of 0.09 is analogous to the average top-k score of a top-k/w subscription with parameters $k = 9$ and $w = 40000$ for the uniform dataset as shown in Figure 4.7a, and these subscriptions thus have similar numbers of matching publications for these values of parameters.

As explained in Section 6.4.1, the number of exchanged messages in the D-ZaLaPS system is the sum of the following:

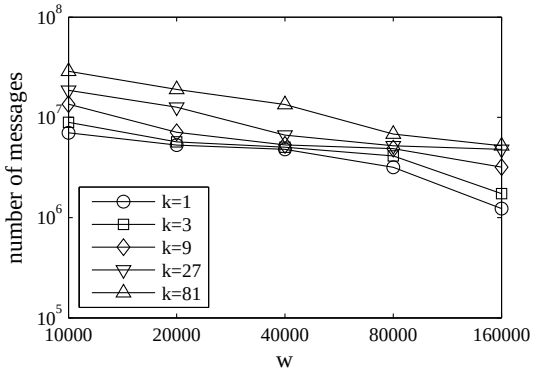
- the number of *subscribe messages* sent by subscriber to rendezvous peers,
- the number of *notify rendezvous peer messages* sent by publisher to rendezvous peers,
- the number of *notify subscriber peer messages* sent by rendezvous to subscriber peers,
- the number of *notify interested peer messages* sent by rendezvous to other interested (rendezvous) peers, and
- the number of *merger update messages* sent by interested rendezvous peers to other rendezvous peers.

The last type of messages is needed only for top-k/w subscriptions and does not appear in D-ZaLaPS with Boolean subscriptions.

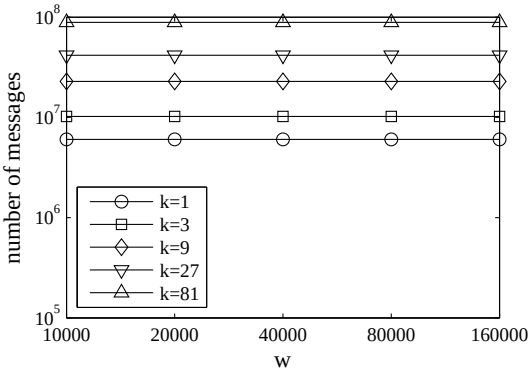
We know the number of subscribe and notify rendezvous peer messages from the default setup shown in Table 6.1, i.e. they are equal to $m = 400$ and $N = 1,000,000$, respectively. Additionally, the number of notify subscriber peer messages equals the number of matching publications shown in Figures 4.8a and 4.8d. Hereafter we show and analyze the number of notify interested peer messages and merger update messages.

Figure 6.21 shows the number of generated notify interested peer messages for different types of subscriptions. The number of such messages directly depends on the subscription threshold values since a subscription with a larger threshold is interested in the larger number of cells. We can see that for larger values of parameter $k$, PA-based subscriptions generate significantly less interested peer messages than RAPF-based subscriptions. This is expected since PA-based subscriptions usually have lower thresholds than the RAPF-based. Also, the number of such messages decreases for PA when increasing $w$, while it remains constant for RAPF. If we compare figures of top-k/w and Boolean subscriptions, we can see that an ideal Boolean subscription with threshold value 0.09 generates only slightly less messages that the corresponding PA-based top-k/w subscription ($k = 9$ and $w = 40000$). However, this is not the case for the RAPF-based subscriptions since they generate significantly more messages than the ideal Boolean subscriptions.

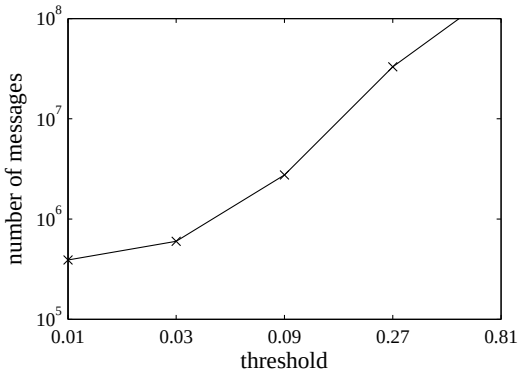
Figure 6.22 shows the number of merger update messages for different types of subscriptions. The number of such messages directly depends on the frequency of subscription threshold and cell changes




(a) PA-based top-k/w subscriptions




(b) RAPF-based top-k/w subscriptions



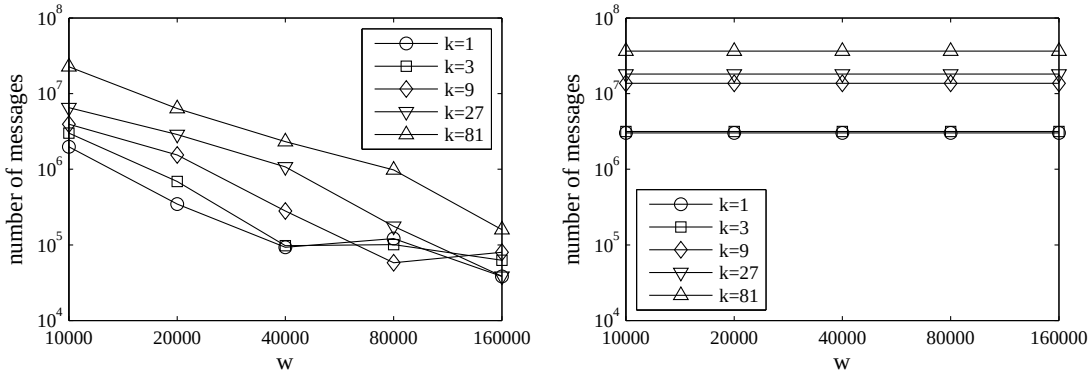

(c) Boolean subscriptions

Figure 6.21: The number of notify interested peer messages in D-ZaLaPS for different types of subscriptions.

shown in Figure 5.9. We can see that PA-based subscriptions generate significantly less merger update messages than the RAPF-based since their thresholds change less frequently than those of the RAPF-based subscriptions. This is expected since the threshold of a RAPF-based subscription is defined as a score of the filter element with rank $k$, and thus it changes more often than the threshold of a PA-based subscription, which is defined as a score[10]of the candidate tree tail. Due to the fact that Boolean subscriptions are static, i.e. have static threshold, this type of messages does not appear in the case of Boolean subscriptions.

Figure 6.23 shows the total number of exchanged messages in the D-ZaLaPS system for different types of subscriptions. We can see that PA-based subscriptions generate significantly less messages than the RAPF-based, and just a slightly more messages than analogous Boolean subscriptions. For example, PA-based subscription generates $6,741,337$ messages for $k = 9$, while its Boolean analogue generates $3,865,667$ messages. The total number of messages is acceptable if we recall that $1,000,000$ publications have been processed. In a naive scenario that employs publication flooding as a routing strategy such that all peers would process all published publications to match them against subscriptions under their

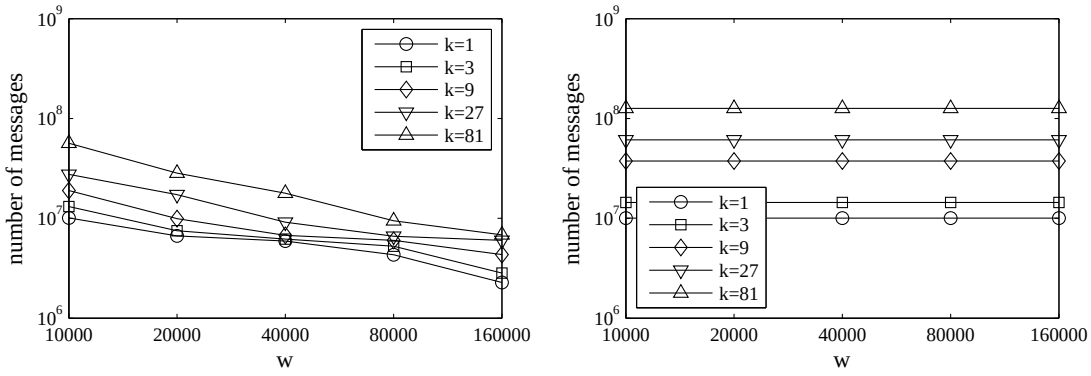
[10]When the threshold is not approximated.




(a) PA-based top-k/w subscriptions

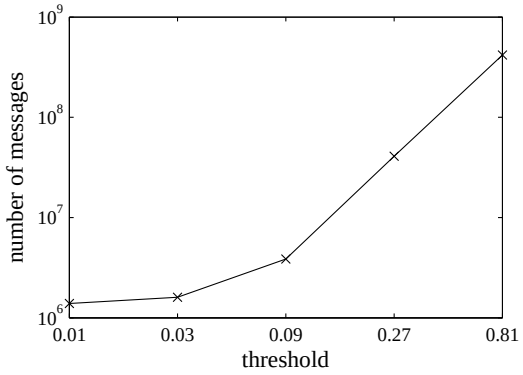
(b) RAPF-based top-k/w subscriptions

Figure 6.22: The number of merger update messages in D-ZaLaPS for different types of top-k/w subscriptions.




(a) PA-based top-k/w subscriptions

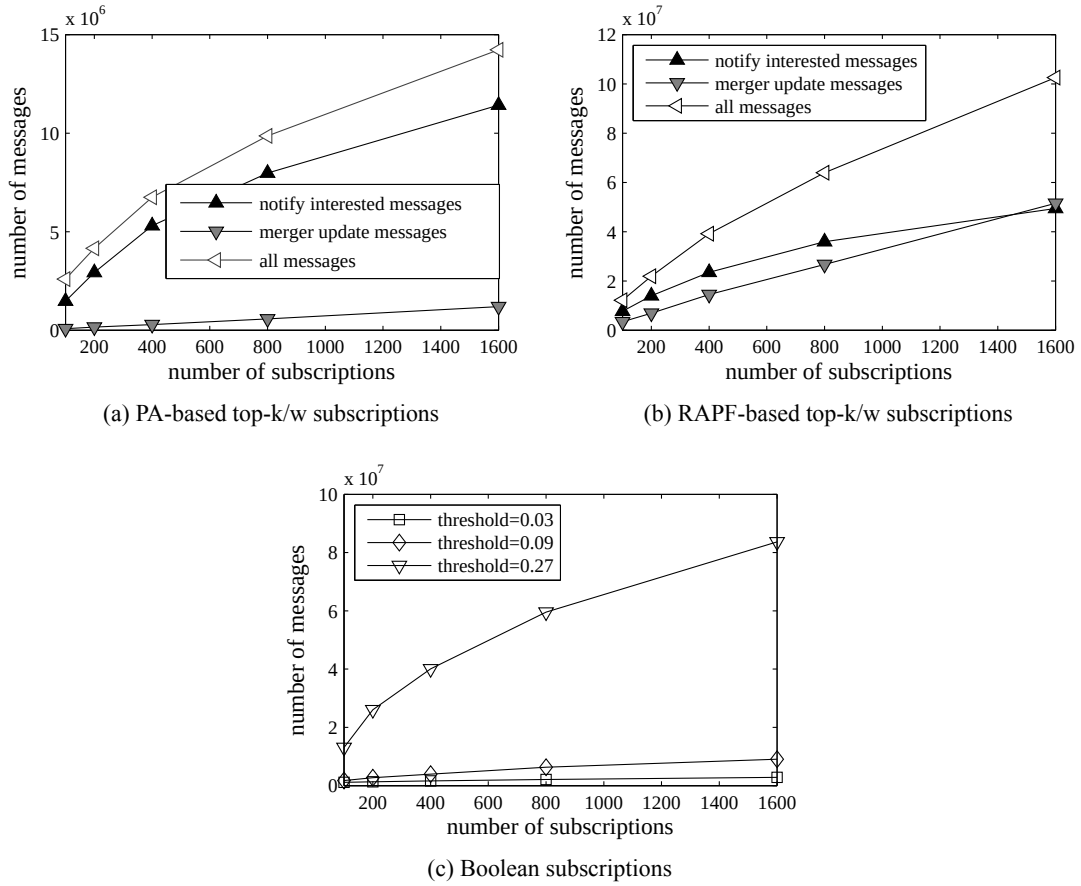
(b) RAPF-based top-k/w subscriptions




(c) Boolean subscriptions

Figure 6.23: Total number of exchanged messages in D-ZaLaPS for different types of subscriptions.

responsibility, the total number of messages exchanged by the peers could climb up to $(C-1)\cdot N = 2.55\times 10^8$ messages.




(a) PA-based top-k/w subscriptions

(b) RAPF-based top-k/w subscriptions

(c) Boolean subscriptions

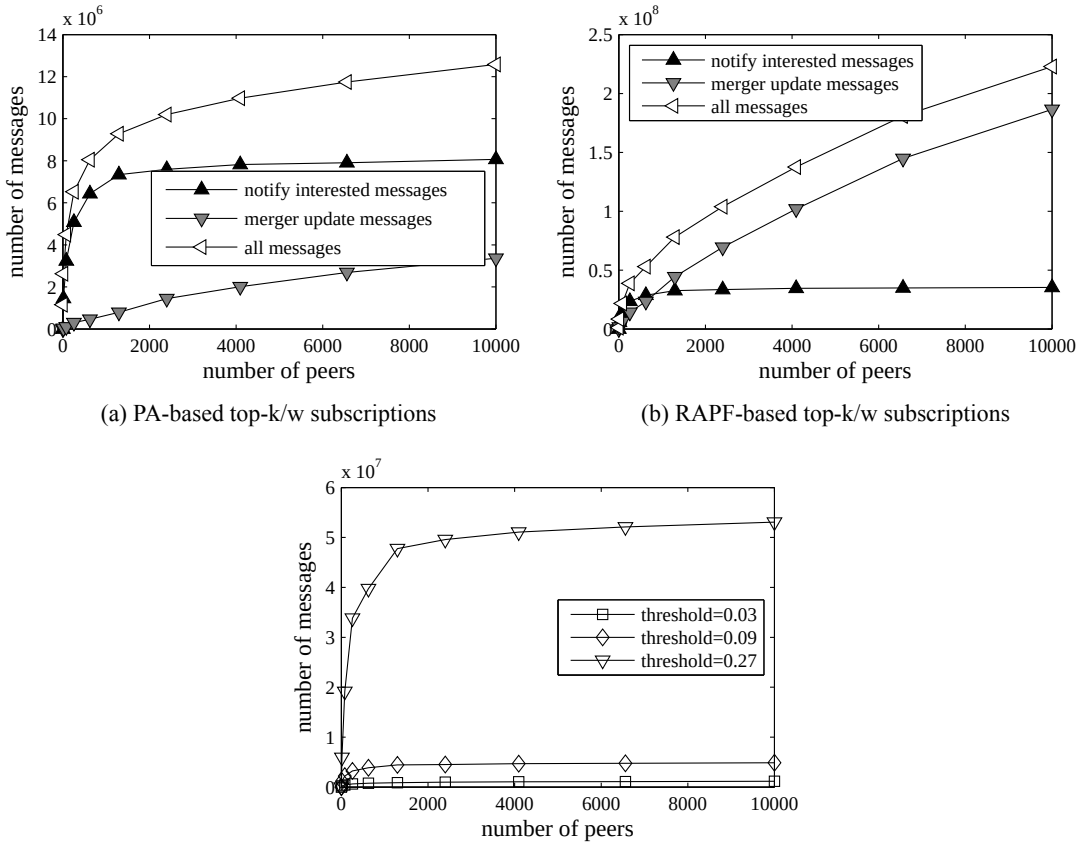
Figure 6.24: Subscription scalability of D-ZaLaPS for different types of subscriptions.

### 6.5.2 Subscription Scalability

In the next experiment we evaluate the scalability of D-ZaLaPS when the number of subscriptions increases. In Figure 6.24 we see that D-ZaLaPS is scalable for all three types of subscriptions since the number of messages increases sub-linearly with the increasing number of subscriptions. However, please note that the number of merger update messages for top-k/w subscriptions scales worse than the number of notify interested peer messages and grows linearly with the increasing number of subscriptions.

### 6.5.3 Peer Scalability

In the next experiment we evaluate the scalability of D-ZaLaPS when the number of peers increase. In Figure 6.25 we see that D-ZaLaPS is scalable for all three types of subscriptions since the number of messages increases logarithmically for Boolean and PA-based top-k/w subscriptions with the increasing number of peers, and sub-linearly for RAPF-based subscriptions. As in the previous experiment, the number of merger update messages for top-k/w subscriptions again scales worse than the number of notify interested peer messages.




(a) PA-based top-k/w subscriptions

(b) RAPF-based top-k/w subscriptions

(c) Boolean subscriptions

Figure 6.25: Peer scalability of D-ZaLaPS for different types of subscriptions.

### 6.5.4 Conclusion

Hereafter, we conclude the presented experimental evaluation. D-ZaLaPS is a scalable distributed top-k/w publish/subscribe system since the number of generated messages increases sub-linearly and logarithmically with increasing number of subscriptions and peers, respectively. In such a distributed scenario with multiple data sources, it is highly probable that publications being processed by a peer follow the random-order data stream model, and therefore PA-based processing is applicable in such setting. D-ZaLaPS generates significantly less messages in the case when PA-based top-k/w subscriptions are used than in the case with RAPF-based subscriptions, mostly due to the difference in the numbers of generated notify interested peer messages, which directly depend on the subscription thresholds. For deterministic top-k/w processing algorithms (i.e. SASF, SARF, SAPF, RASF, RARF and RAPF), a subscription threshold is defined as the score of the filter element with rank $k$, in the case of SF and RF, or as the filter threshold in the case of PF, and the results thus would be similar to the presented if any other deterministic top-k/w processing algorithm was used instead of RAPF. Finally, analogous to the centralized top-k/w system with selective routing, D-ZaLaPS also generates less messages in the case of PA-based top-k/w subscriptions than in the case of inadequately defined Boolean subscriptions.

## 6.6 Related Work

Large-scale publish/subscribe systems are regularly implemented as distributed systems due to scalability problems of centralized solutions. For example, [29] shows that either of the existing centralized enterprise service bus (ESB) products on the market sustains a high throughput when the number of subscribers increases. For this reason, distributed publish/subscribe systems have attracted a lot of attention from the research community in the last 15 years. However, despite extensive research efforts done in this period, large-scale publish/subscribe systems are yet to be widely deployed [175]. In this section we present a brief overview of distributed publish/subscribe systems in which special attention will be given to the used routing strategies, i.e. we will focus on the routing layer in the referent architectural model shown in Figure 6.1. A more detailed overview would be out of scope of this thesis and thus we refer the interested reader to [147, 22].

### 6.6.1 Distributed Publish/Subscribe Systems with Flooding

Publication flooding as a routing strategy in distributed publish/subscribed systems can be very easily implemented and thus was used in the first publish/subscribe prototypes [155]. However, this strategy is very rarely used since it does not scale in terms of communication overhead. In literature, it mainly serves as a referent routing strategy in experimental evaluations, e.g. [44], JEDI [69] and TERA [19], but sometimes is also used in implementations, e.g. Gryphoon [24] and [37].

Subscription flooding, as the opposite routing strategy, generates a very high communication overhead when subscriptions change at a high rate [186, 46, 144], and thus is also rarely used in distributed publish/subscribe systems. However, since subscribers can be reached in a single hop and non-matching publications can be filtered at the publisher side when this strategy is used, some of more recent distributed publish/subscribe systems still apply it, e.g. Gryphoon [24], RUBDES [184] and MEDYM [41].

### 6.6.2 Distributed Publish/Subscribe Systems with Selective Routing

Selective routing strategies (i.e. covering-based and rendezvous routing) are the most popular routing strategies in distributed publish/subscribe systems. Since these strategies are deterministic and do not flood the overlay network with messages, they are much more interesting for the research community than gossiping and flooding strategies, respectively.

Covering-based routing is first introduced in SIENA[46, 47], and after that has been applied in many distributed publish/subscribe systems such as JEDI [69], REBECA [145, 144], PADRES [126, 125], etc. This routing strategy has attracted a lot of attention of the research community which has resulted in many different improvements of this routing strategy, e.g. [196, 159, 188, 28, 20, 127]. Since we identified this routing strategy as not well suited for top-k/w subscriptions, we will not discuss it further.

Contrary to covering-based routing, we identified rendezvous routing as a well suited routing strategy for distributed top-k/w systems. This routing strategy is proposed in Scribe [182, 49] and is used in many other distributed publish/subscribe systems such as Bayeux [209], Hermes [165], Kyra [40], Meghdoot [102], Reach [163], HOMED [62], Ferry [207], etc. Scribe [182, 49] is built on top of Pastry [181],

a generic peer-to-peer object location and routing scheme for a large-scale overlay network of nodes connected via the Internet. It supports only topic-based subscriptions and is based on a simple idea to map each of the available topics to a rendezvous Pastry node. Every new subscription and every newly published publication are routed by Pastry to the corresponding rendezvous Pastry node. For each topic, Scribe maintains a separate multicast tree rooted at the topic rendezvous node, and use it to disseminate publications published on that topic. Bayeux [209] is very similar to Scribe, since it supports multiple topics and also maintains a multicast tree per topic. However, it is built on top of Tapestry [205], a wide-area location and routing architecture, and has a few scalability problems when compared to Scribe [182]. Hermes [165] supports type-based subscriptions. Similarly to Siena, Hermes builds a diffusion tree for each type, which is rooted at the corresponding rendezvous node. Again, subscriptions and publications of the same type are routed to the rendezvous node for this type. When a new publication in published, the corresponding diffusion tree is flooded with it, and then every subscriber of this type performs matching for its subscriptions. Therefore, Hermes uses a combination of rendezvous routing and publication flooding. On the other side, Kyra [40] uses a hybrid routing scheme which is a combination of rendezvous routing and filtering-based routing, where the latter is a simpler form of the covering-based routing. In this system, the attribute space is partitioned in non-overlapping zones first among groups of nodes (cliques) and again among nodes in each clique. In this way, each clique has a rendezvous node for every part of the attribute space. Additionally, rendezvous nodes that are responsible for the same parts of the attribute space in different cliques are connected together in a routing tree. Then, rendezvous routing is used between nodes in each clique, and filtering-based routing between nodes in each routing tree. Meghdoot [102] is a distributed publish/subscribe system with content-based subscriptions that is based on pure rendezvous routing. This system uses only the CAN [178] structured overlay for communication between nodes and does not create separate trees for this purpose. The CAN assigns non-overlapping zones of the attribute space to nodes in the overlay network such that each node is connected to nodes with neighboring zones. This way CAN ensures that each part of the attribute space has a responsible node, which allows routing messages from a point to any other point in the attribute space. Chandramouli at al. [55] present a distributed content-based system with rendezvous routing that is based on CAN as Meghdoot. Instead of the CAN overlay network, Terpstra at al. [197] use Chord [194] and algorithms for subscription covering and merging from Rebeca [149] to build a distributed content-based system that uses a hybrid of rendezvous and covering-based routing. Building a distributed content-based system on top of a distributed topic-based system with rendezvous routing is also considered in literature, e.g. Tam at al. [195] present a distributed content-based system which is built on top of Scribe.

### 6.6.3 Distributed Publish/Subscribe Systems with Gossiping

Gossiping routing strategies (i.e. basic and informed gossiping) are also very popular in publish/subscribe research community since they lead to high reliability, robustness and self stabilization [30, 182, 23, 22]. Examples of distributed publish/subscribe systems which employ these routing strategies are Costa at al. [66, 65], SpiderCast [61], Tera [19], pmcast [84] and Sub-2-Sub [198]. Costa at al. [65, 66] present a content-based publish/subscribe system with three different approaches to the basic gossiping: publisher-

based push, publisher-based pull and subscriber-based pull. The authors also experimentally evaluate these three approaches in [66]. All other mentioned systems employ informed gossiping as a routing strategy. To accomplish the deterministic part of routing, these systems either cluster nodes with similar interests in additional overlay trees on top of the basic overlay (Tera and Sub-2-Sub) or build a single overlay in which nodes with similar interest are kept close to each other (SpiderCast and pmcast). It is important to mention that SpiderCast and Tera support only topic-based subscriptions, whereas Sub-2-Sub and pmcast support content-based subscriptions.

### 6.6.4 Other Relevant Work

The work most relevant to ours is Huang at al. [111]. In this work, a new type of stateful subscription is introduced in publish/subscribe systems, which is called the *parametrized subscription*. Our top-k/w proxy subscriptions are parametrized subscriptions with the subscription threshold as the only parameter. The authors discuss two cases, one centralized and one distributed with filtering-based routing. In both cases, every change of a parameter is published as a publication on this parameter topic. Their conclusion is that every parameter change has to be propagated to all brokers in the distributed case. Please note that we have reached the same conclusion for covering-based routing as an advanced version of filtering-based routing.

# CHAPTER 7

# Conclusions and Future Work

Even though extensive research efforts have been done in the last 15 years in the area of publish/subscribe systems, wide-area publish/subscribe systems are not yet widely deployed. According to [175], the main reasons for their slow acceptance are the complexity of the general matching problem, system heterogeneity, and lack of wide-area deployments which delays development of improved, real-life usable solutions.

In this thesis we argued that there is another reason for the lack of adoption of such systems—an unpredictable number of matching publications delivered to subscribers with the currently prevailing Boolean matching model. The Boolean matching model may often deliver either too many or too few publications, and this may cause user dissatisfaction with the provided service. In general, a user percepts the entire system through both the quantity and quality of received publications. Therefore, a large quantity of received publications will be considered as a sort of spam, while a system that delivers too few publications might be recognized as non-working. The number of received publications is crucial for the acceptance of an actual system by users, even more if, for example, subscribes pay for each delivered publication matching subscriber information interests.

In this thesis we proposed a new publish/subscribe matching model, named the *top-k/w matching model*. This matching model enables a subscriber to control the number of publications it will receive per subscription within a predefined time period. In this matching model, each subscription in a system defines an arbitrary and time-independent scoring function and the parameters $k \in \mathbb{N}$ and $w \in \mathbb{R}^+$. In contrast to the Boolean matching model, this matching model is time-dependent because at each point in time $t$, the parameter $k$ limits the number of matching publications restricting it to the $k$ best scored publications that are published between points in time $t - w$ and $t$.

We consider this model quite intuitive for Internet users as it is requires that each subscriber defines a scoring function and parameters $k$ and $w$. Additionally, the scoring function can be defined by application developers for the entire system which enables subscribers to define simple queries similar to search engine queries. Typical users are already familiar with the concepts of query, ranking and top-$k$ results since

every web search engine works in this way. Moreover, the additional parameter *w*, as the subscription window size, is quite intuitive as well.

In this thesis we showed that publish/subscribe systems based on the top-k/w matching model can be implemented in an efficient way both having centralized and distributed architecture. For each of the commonly used routing strategies in (both centralized and distributed) Boolean publish/subscribe systems we proposed the changes necessary to apply this strategy in top-k/w systems. We also showed that 2 out of 3 centralized and 4 out of 6 distributed routing strategies are particularly well suited for top-k/w systems since they introduce insignificant additional communication overhead in these systems when compared to the equivalent Boolean systems.

## 7.1 Contributions

### 7.1.1 Formal Specification of Boolean and Top-k/w Publish/Subscribe Systems

In this thesis we formally defined the top-k/w matching model with time-based and number-based sliding-windows in the Metric Interval Temporal Logic (MITL) [13] and set theory [34], respectively. We also presented formal specifications of the top-k/w publish/subscribe system with time-based windows and Boolean publish/subscribe system that are based on timed words (i.e. sequential traces of timed events) and use the syntax of MITL. Both specifications define several safety and liveness properties which are standard means for specifying real-time system behavior. In the case of publish/subscribe systems, the safety property asserts that a non-matching publication will never be delivered to subscriber, while liveness property asserts that every matching publication will eventually (i.e. after the propagation of the corresponding subscription) be delivered to subscriber.

Although several formal specifications of Boolean publish/subscribe systems in temporal logic exist in literature, we have provided a new specification since existing specifications are not general enough, i.e. they do not cover all possible usage cases (e.g. receiving matching publications and canceling the corresponding subscription afterwards) and do not define finite processing delays in a system, which is extremely important for the quality of service.

We compared our specifications of Boolean and top-k/w systems, and showed that every Boolean system is a special case of the corresponding top-k/w system. As a consequence, every practical implementation of the top-k/w system model automatically supports the behavior of a Boolean system without any further modifications. This allows us to have Boolean and top-k/w subscriptions simultaneously active in the same top-k/w system, which is very important for the future acceptance of the top-k/w matching model.

### 7.1.2 Top-k/w Processing over Data Streams

A sliding-window top-k (*top-k/w*) query monitors incoming data stream objects within a sliding window to identify k best-ranked objects with respect to an arbitrary scoring function. Efficient processing of such queries is challenging because, even though an object is not a top-k object at the time when it enters the

processing system, it might become one in future, and therefore a set of potential top-k objects within the sliding window has to be stored in memory. Existing solutions either assume processing environments with unbounded memory, or scale poorly for large values of parameter k and window size. Moreover, they are typically tailored to specific scoring functions, and thus in this thesis we defined a generic top-k/w processing model that is independent of data representation and scoring functions.

We also proposed several novel algorithms for efficient top-k/w processing of multiple queries in resource-constrained environments publishing data at high rates. First, we proposed the Relaxed Candidate Pruning Algorithm (RA), a novel deterministic algorithm which, instead of maintaining the minimal set of potential top-k objects in a k-skyband, periodically prunes the k-skyband to offer improved processing performance at the expense of an increased memory consumption. Furthermore, we proposed the Probabilistic Candidate Pruning Algorithm (PA), a probabilistic algorithm for random-order data streams which surpasses deterministic algorithms in terms of memory consumption and processing performance, at the expense of producing approximate answers to top-k/w queries. To further improve deterministic algorithm implementations, we introduced a PA-based buffer of most recent data objects. Our complexity analysis and extensive comparative evaluation using both synthetic and real datasets showed that the proposed algorithms significantly outperform the competing approaches.

### 7.1.3 Centralized and Distributed Top-k/w Publish/Subscribe Systems

Since scalability is an essential characteristic of publish/subscribe systems, we gave special attention to design algorithms supporting the top-k/w matching model which target large-scale publish/subscribe systems. For each of the commonly used routing strategies in both centralized and distributed publish/subscribe systems, we first proposed the changes necessary to apply this strategy in a top-k/w system, then we identified locations of top-k/w processors and recent buffers in the system, and finally we explained how subscriptions and publications are routed from clients to these processors.

Our analysis showed that subscription flooding as a routing strategy in both centralized and distributed publish/subscribe systems is not well suited for top-k/w systems since the synchronization of subscriptions and their copies introduces a large communication overhead in top-k/w systems when compared to equivalent centralized Boolean systems. Additionally, the analysis showed that, for the same reason, this is also true for covering-based routing in distributed top-k/w systems. Finally, the analysis showed that other routing strategies (i.e. selective routing for centralized systems, publication flooding for both centralized and distributed systems, and rendezvous routing, basic gossiping and informed gossiping for distributed systems) are particularly well suited for top-k/w systems since they do not introduce any significant additional communication overhead when compared to the equivalent Boolean systems. Additionally, we presented and experimentally evaluated D-ZaLaPS, a distributed top-k/w publish/subscribe system which is built on top of CAN [178]. The experimental evaluation shows that D-ZaLaPS is scalable both for increasing number of peers and subscriptions.

In this thesis we presented the results of an experimental evaluation of the Boolean and top-k/w systems using sliding-window k-NN subscriptions (i.e. queries) as a case study. This experimental evaluation shows that, assuming subscribers are interested in only top-k publications in the sliding window,

the top-k/w systems have lower message overhead when compared to the equivalent Boolean systems in which subscriptions are inadequately defined.

## 7.2 Future Work

There are several additional aspects related to top-k/w publish/subscribe systems that can be addressed by future work.

**Mobility-enabled top-k/w system.** In this thesis we assumed static environments where both publishers and subscribers are stationary. However, in a large-scale publish/subscribe system, it is expected that many clients will probably use mobile devices. Since such devices are not connected to the network all the time, it is very important to investigate specific problems that appear in dynamic environments related to content caching and delivery. Additionally, these devices have quite heterogeneous capabilities, which is an additional problem in this setting.

**Other approaches to publication ranking.** As we have shown in this thesis, the problem with Boolean subscriptions is the sharp boundary of the subscription subspace of interest. Similarly, top-k/w subscriptions define such a sharp boundary in time since publications are instantly invalidated when dropped from the subscription window. An interesting future research problem is to see how publication scores can gradually decrease in time. For example, publication freshness defined in [75] is one of the possible approaches to this problem.

**Supporting other scoring functions in D-ZaLaPS.** The presented system D-ZaLaPS supports only distance scoring functions. Since this is just one of the possible types of scoring functions, the next logical step is support other types of scoring functions, such as aggregation and relevance functions.

**Indexing top-k/w subscriptions in vector-space.** An indexing technique specifically designed to process top-k/w subscriptions that define relevance scoring functions in vector-space model for unstructured data upon arrival of a new publication is an open research area. To our knowledge, the only work which studies this problem is [142]. Since the collection (of publications) is highly dynamic in this case, which is contrary to the current assumption in information retrieval, it is very important to see how inverse document frequency (IDF) can be defined for such a dynamic collection. This type of indexing is very important and might be a prerequisite for efficient implementation of a distributed top-k/w system supporting relevance scoring functions.

**Distributed top-k/w system under churn.** D-ZaLaPS do not address the dynamics of the underlying peer-to-peer overlay network and therefore an important aspect of the future work would be to test its performance under churn, which is a more realistic setting. Additionally, similarly to distributed Boolean systems, informed gossiping could be better a routing strategy for such a setting, and thus should be further examined.

# Bibliography


[1] D. J. Abadi, D. Carney, U. Tintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12:120–139, 2003.

[2] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *ICALP*, 1989.

[3] S. Acharya, M. J. Franklin, and S. B. Zdonik. Disseminating updates on broadcast disks. In *VLDB*, 1996.

[4] C. Aggarwal. *Data Streams – Models and Algorithms*. Springer, 2007.

[5] C. C. Aggarwal. *Data Streams: Models and Algorithms (Advances in Database Systems)*. Springer-Verlag New York, Inc., 2006.

[6] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC*, 1999.

[7] K. Akkaya and I. Ari. In-network data aggregation in wireless sensor networks. In *The Handbook of Computer Networks*. John-Wiley & Sons, 2007.

[8] G. Alandjani and E. Johnson. Fuzzy routing in ad hoc networks. In *IPCCC*, 2003.

[9] B. Alpern and F. B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21:181–185, 1985.

[10] M. Altherr, M. Erzberger, and S. Maffeis. ibus-a software bus middleware for the java platform. In *SRDS*, 1999.

[11] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[12] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[13] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43:116–146, 1996.

[14] R. Alur and T. A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *FOCS*, 1992.

[15] R. Alur and T. A. Henzinger. Real-time logics: complexity and expressiveness. *Inf. Comput.*, 104:35–77, 1993.

[16] R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41:181–203, 1994.

[17] M. Balazinska, A. Deshpande, M. J. Franklin, P. B. Gibbons, J. Gray, M. Hansen, M. Liebhold, S. Nath, A. Szalay, and V. Tao. Data management in the worldwide sensor web. *IEEE Pervas. Comput.*, 6:30–40, 2007.

[18] R. Baldoni, R. Beraldi, S. T. Piergiovanni, and A. Virgillito. On the modelling of publish/subscribe communication systems: Research articles. *Concurr. Comput. : Pract. Exper.*, 17:1471–1495, 2005.

[19] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *DEBS*, 2007.

[20] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Efficient publish/subscribe through a self-organizing broker overlay and its application to siena. *Comput. J.*, 50:444–459, 2007.

[21] R. Baldoni, M. Contenti, S. T. Piergiovanni, and A. Virgillito. Modelling publish/subscribe communication systems: Towards a formal approach. In *WORDS*, 2003.

[22] R. Baldoni, L. Querzoni, S. Tarkoma, and A. Virgillito. Distributed event routing in publish/subscribe systems. In *MiNEMA*, 2009.

[23] R. Baldoni and A. Virgillito. Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey (revised version). Technical report, Universitá di Roma la Sapienza, 2006.

[24] G. Banavar, T. Chandra, B. M. andJay Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *ICDCS*, 1999.

[25] H. Barringer, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds. *The imperative future: principles of executable temporal logic*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[26] T. A. H. Bell and A. Moffat. The design of a high performance information filtering system. In *SIGIR*, 1996.

[27] M. K. Bergman. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1):online, 2001.

[28] S. Bianchi, P. Felber, and M. Gradinariu. Content-based publish/subscribe using distributed r-trees. *Lecture Notes in Computer Science*, 4641:537–548, 2007.

[29] K. Birman, J. Cantwell, D. Freedman, Q. Huang, P. Nikolov, and K. Ostrowski. Edge mashups for service-oriented collaboration. *Computer*, 42:90–94, 2009.

[30] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17:41–88, 1999.

[31] A. Blair. Reading strategies for coping with information overload ca. 1550-1700. *Journal of the History of Ideas*, 64:11–28, 2003.

[32] C. Böhm, B. C. Ooi, C. Plant, and Y. Yan. Efficiently processing continuous k-nn queries on data streams. In *ICDE*, 2007.

[33] K. D. Bollacker, S. Lawrence, and C. L. Giles. Discovering relevant scientific literature on the web. *IEEE Intell. Syst.*, 15:42–47, 2000.

[34] I. Bronshtein, K. Semendyayev, G. Musiol, and H. Mühlig. *Handbook of Mathematics*. Springer-Verlag, 2007.

[35] F. Bruss. Sum the odds to one and stop. *Annals of Probability*, 28(3):1384–1391, 2000.

[36] F. Bruss and D. Paindaveine. Selecting a sequence of last successes in independent trials. *Journal of Applied Probability*, 37:389–399, 2000.

[37] G. Buchanan and A. Hinze. A generic alerting service for digital libraries. In *JCDL*, 2005.

[38] J. But, G. Armitage, and L. Stewart. Outsourcing automated qos control of home routers for a better online game experience. *Communications Magazine, IEEE*, 46:64–70, 2008.

[39] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In *ICSE*, 2001.

[40] F. Cao and J. P. Singh. Efficient event routing in content-based publish/subscribe service network. In *INFOCOM*, 2004.

[41] F. Cao and J. P. Singh. Medym: Match-early and dynamic multicast for content-based publish-subscribe service networks. In *ICDCSW*, 2005.

[42] M. Caporuscio and P. Inverardi. Uncertain event-based model for egocentric context sensing. In *SEM*, 2005.

[43] A. M. Carlo A. Furia, Dino Mandrioli and M. Rossi. Modeling time in computing: a taxonomy and a comparative survey. Technical report, Politecnico di Milano, 2007.

[44] A. Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, 1998.

[45] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *PODC*, 2000.

[46] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19:332–383, 2001.

[47] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *INFOCOM*, 2004.

[48] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *SIGCOMM*, 2003.

[49] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE Journal on Selected Areas in Communications*, 2002.

[50] A. Chakrabarti, G. Cormode, and A. McGregor. Robust lower bounds for communication and stream computation. In *STOC*, 2008.

[51] A. Chakrabarti, T. S. Jayram, and M. Pătraşcu. Tight lower bounds for selection in randomly ordered streams. In *SODA*, 2008.

[52] S. Chakravarthy and Q. Jiang. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*. Springer Publishing Company, Incorporated, 2009.

[53] R. Chand and P. Felber. Xnet: A reliable content-based publish/subscribe system. In *SRDS*, 2004.

[54] R. Chand and P. Felber. Semantic peer-to-peer overlays for publish/subscribe networks. *Lecture Notes in Computer Science*, 3648:1194–1204, 2005.

[55] B. Chandramouli, J. Xie, and J. Yang. On the database/network interface in large-scale publish/subscribe systems. In *SIGMOD*, 2006.

[56] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: continuous dataflow processing. In *SIGMOD*, 2003.

[57] D. Chatziantoniou and G. Doukidis. *Encyclopedia of Information Science and Technology*, chapter Data Streams as an Element of Modern Decision Support, pages 941–949. Information Resources Management Association, 2009.

[58] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagaracq: a scalable continuous query system for internet databases. *SIGMOD Rec.*, 29:379–390, 2000.

[59] E. G. Chewning and A. M. Harrell. The effect of information load on decision makers' cue utilization levels and decision quality in a financial distress decision task. *Accounting, Organizations and Society*, 15:527 – 542, 1990.

[60] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33:427–469, 2001.

[61] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *DEBS*, 2007.

[62] Y. Choi, K. Park, and D. Park. Homed: A peer-to-peer overlay architecture for large-scale content-based publish/subscribe systems. In *DEBS*, 2004.

[63] M. Cilia. *An Active Functionality Service for Open Distributed Heterogeneous Environments*. PhD thesis, Darmstadt University of Technology, 2002.

[64] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[65] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola. Introducing reliability in content-based publish-subscribe through epidemic algorithms. In *DEBS*, 2003.

[66] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola. Epidemic algorithms for reliable content-based publish-subscribe: An evaluation. In *ICDCS*, 2004.

[67] S. Courtenage. Specifying and detecting composite events in content-based publish/subscribe systems. In *ICDCSW*, 2002.

[68] G. Cugola and J. E. M. de Cote. On introducing location awareness in publish-subscribe middleware. In *ICDCSW*, 2005.

[69] G. Cugola, E. D. Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE T. Software. Eng.*, 27:827–850, 2001.

[70] K. S. D. Souravlias, M. Drosou and E. Pitoura. On novelty in publish/subscribe delivery. In *DBRank*, 2010.

[71] G. Das, D. Gunopulos, N. Koudas, and N. Sarkas. Ad-hoc top-k query answering for data streams. In *VLDB*, 2007.

[72] A. K. Datta, M. Gradinariu, M. Raynal, and G. Simon. Anonymous publish/subscribe in p2p networks. In *IPDPS*, 2003.

[73] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In *EDBT*, 2006.

[74] Y. Diao, P. Fischer, M. Franklin, and R. To. Yfilter: Efficient and scalable filtering of xml documents. In *ICDE*, 2002.

[75] M. Drosou. Ranked publish/subscribe delivery. In *DEBS PhD*, 2009.

[76] M. Drosou, E. Pitoura, and K. Stefanidis. Preferential publish/subscribe. In *PersDB*, 2008.

[77] M. Drosou, K. Stefanidis, and E. Pitoura. Preference-aware publish/subscribe delivery with diversity. In *DEBS*, 2009.

[78] D. D'Souza and P. Prabhakar. On the expressiveness of mtl in the pointwise and continuous semantics. *Int. J. Softw. Tools Technol. Transf.*, 9:1–4, 2007.

[79] Z. M. Edward Chang and A. Pnueli. *Logic and Algebra of Specifications*, chapter The Safety-Progress Classification. Springer-Verlag, 1991.

[80] M. J. Eppler and J. Mengis. The concept of information overload: A review of literature from organization science, accounting, marketing, mis, and related disciplines. *Inf. Soc.*, 20:325–344, 2004.

[81] P. Eugster. *Type-Based Publish/Subscribe*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2001.

[82] P. Eugster. Type-based publish/subscribe: Concepts and experiences. *ACM Trans. Program. Lang. Syst.*, 29:6, 2007.

[83] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35:114–131, 2003.

[84] P. T. Eugster and R. Guerraoui. Probabilistic multicast. In *DSN*, 2002.

[85] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD*, 2001.

[86] W. Fan, Y. an Huang, H. Wang, and P. S. Yu. Active mining of data streams. In *SDM*, 2004.

[87] A. Farroukh, E. Ferzli, N. Tajuddin, and H.-A. Jacobsen. Parallel event processing for content-based publish/subscribe systems. In *DEBS*, 2009.

[88] L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. *Lecture Notes in Computer Science*, 2672:103–122, 2003.

[89] L. Fiege, G. Mühl, and F. C. Gärtner. A modular approach to build structured event-based systems. In *SAC*, 2002.

[90] C. A. Furia and M. Rossi. On the expressiveness of mtl variants over dense time. In *FORMATS*, 2007.

[91] J. Gama and M. Gaber. *Learning from Data Streams – Processing techniques in Sensor Networks*. Springer, 2007.

[92] J. F. Gantz. The expanding digital universe: A forecast of worldwide information growth through 2010. Technical report, International Data Corporation, 2007. http://www.emc.com/collateral/analyst-reports/expanding-digital-idc-white-paper.pdf.

[93] J. F. Gantz. The diverse and exploding digital universe: An updated forecast of worldwide information growth through 2011. Technical report, International Data Corporation, 2008. http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf.

[94] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. Spade: the system s declarative stream processing engine. In *SIGMOD*, 2008.

[95] D. Geer. Is it really time for real-time search? *Computer*, 43:16–19, 2010.

[96] V. Goebel and T. Plagemann. Data stream management systems - a technology for network monitoring and traffic analysis? In *ConTEL*, 2005.

[97] L. Golab. *Sliding window query processing over data streams*. PhD thesis, University of Waterloo, 2006.

[98] L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Rec.*, 32:5–14, 2003.

[99] J. Gough and G. Smith. Efficient recognition of events in a distributed system. *Australian computer science communications*, 17:173–179, 1995.

[100] R. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the ready event notification service. *ICDCS*, 00:0108, 1999.

[101] S. Guha and A. McGregor. Approximate quantiles and the order of the stream. In *PODS*, 2006.

[102] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. *Lecture Notes in Computer Science*, 3231:254–273, 2004.

[103] P. Haghani, S. Michel, and K. Aberer. Evaluating top-k queries over incomplete data streams. In *CIKM*, 2009.

[104] D. Halperin. Arrangements. In *Handbook of Discrete and Computational Geometry*. CRC Press LLC, 2004.

[105] C.-S. Han, S. K. Lee, and M. England. Transition to postmodern science—related scientometric data. *Scientometrics*, 84:391–401, 2009.

[106] E. N. Hanson, M. Chaabouni, C.-H. Kim, and Y.-W. Wang. A predicate matching algorithm for database rule systems. In *SIGMOD*, 1990.

[107] M. Hapner, R. Burridge, and R. Sharma. Java message service specification - version 1.1. Technical report, Sun Microsystems, 2002. http://java.sun.com/products/jms/docs.html.

[108] T. A. Henzinger. *The temporal specification and verification of real-time systems*. PhD thesis, Stanford University, 1992.

[109] T. A. Henzinger. It's about time: Real-time logics reviewed. In *CONCUR*, 1998.

[110] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *ICALP*, 1992.

[111] Y. Huang and H. Garcia-Molina. Parameterized subscriptions in publish/subscribe systems. *Data & Knowledge Engineering*, 60:435–450, 2007.

[112] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *KDD*, 2001.

[113] T. Imieliński, S. Viswanathan, and B. R. Badrinath. Power efficient filtering of data on air. In *EDBT*, 1994.

[114] C. Jin, K. Yi, L. Chen, J. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. *The VLDB Journal*, 19:411–435, 2010.

[115] S. Kale, E. Hazan, F. Cao, and J. P. Singh. Analysis and algorithms for content-based event matching. In *ICDCSW*, 2005.

[116] E. Kindler. Safety and liveness properties: A survey. *Bulletin of the European Association for Theoretical Computer Science*, 53:268–272, 1994.

[117] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang. Approximate nn queries on streams with guaranteed error/performance bounds. In *VLDB*, 2004.

[118] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2:255–299, 1990.

[119] F. Kröger and S. Merz. State systems. In *Temporal Logic and State Systems*. Springer Berlin Heidelberg, 2008.

[120] B. Krishnamurthy and D. S. Rosenblum. Yeast: A general purpose event-action system. *IEEE T. Software. Eng.*, 21(10):845–857, 1995.

[121] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3:125–143, 1977.

[122] L. Lamport. Verification and specifications of concurrent programs. *Lecture Notes in Computer Science*, 803:347–374, 1994.

[123] P. O. Larsen and M. von Ins. The rate of growth in scientific publication and the decline in coverage provided by science citation index. *Scientometrics*, 84:575–603, 2010.

[124] A. Lekova, K. Skjelsvik, T. Plagemann, and V. Goebel. Fuzzy logic-based event notification in sparse manets. In *AINAW*, 2007.

[125] G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *ICDCS*, 2005.

[126] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. *Lecture Notes in Computer Science*, 3790:249–269, 2005.

[127] G. Li, V. Muthusamy, and H.-A. Jacobsen. Adaptive content-based routing in general overlay topologies. *Lecture Notes in Computer Science*, 5346:1–21, 2008.

[128] H. Liu and H.-A. Jacobsen. A-topss – a publish/subscribe system supporting imperfect information processing. In *VLDB*, 2004.

[129] H. Liu and H.-A. Jacobsen. Modeling uncertainties in publish/subscribe systems. In *ICDE*, 2004.

[130] C. Low, J. Randell, and M. Wray. Self-describing data representation (sdr). Technical report, Hewlett-Packard Labs, 1997.

[131] X. Lu, X. Li, T. Yang, Z. Liao, W. Liu, and H. Wang. RRPS: A ranked real-time publish/subscribe using adaptive QoS. *Lecture Notes in Computer Science*, 5593:835–850, 2009.

[132] G. Luo, C. Tang, and P. S. Yu. Resource-adaptive real-time new event detection. In *SIGMOD*, 2007.

[133] P. Lyman and H. R. Varian. How much information? Technical report, University of California at Berkeley, 2000. http://www2.sims.berkeley.edu/research/projects/how-much-info/.

[134] P. Lyman and H. R. Varian. How much information 2003? Technical report, University of California at Berkeley, 2003. http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/index.htm.

[135] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram. Scalable ranked publish/subscribe. In *VLDB*, 2008.

[136] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[137] M. Mansouri-Samani and M. Sloman. Gem: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4:96–108, 1997.

[138] R. Meier and V. Cahill. Steam: Event-based middleware for wireless ad hoc network. In *ICDCSW*, 2002.

[139] D. Moreto and M. Endler. Evaluating composite events using shared trees. *IEE Proceedings - Software*, 148:1–10, 2001.

[140] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system. In *CIDR*, 2003.

[141] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*, 2006.

[142] K. Mouratidis and H. Pang. An incremental threshold method for continuous text search queries. In *ICDE*, 2009.

[143] K. Mouratidis and D. Papadias. Continuous nearest neighbor queries over sliding windows. *IEEE Trans. on Knowl. and Data Eng.*, 19:789–803, 2007.

[144] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

[145] G. Mühl and L. Fiege. Supporting covering and merging in Content-Based publish/subscribe systems: Beyond name/value pairs. *IEEE Distributed Systems Online (DSOnline)*, 2:1–7, 2001.

[146] G. Mühl, L. Fiege, and A. P. Buchmann. Filter similarities in content-based publish/subscribe systems. In *ARCS*, 2002.

[147] G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[148] G. Mühl and A. Tanner. A formalisation of message-complete publish/subscribe systems. In *DISC*, 2004.

[149] G. Mühl, A. Ulbrich, K. Herrmann, and T. Weis. Disseminating information to mobile clients using publish-subscribe. *IEEE Internet. Comput.*, 8:46–53, 2004.

[150] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. In *SFCS*, 1978.

[151] J. Muramatsu and W. Pratt. Transparent queries: investigation users' mental models of search engines. In *SIGIR*, 2001.

[152] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. Analysis and evaluation of v*-knn: an efficient algorithm for moving knn queries. *The VLDB Journal*, 19:307–332, 2010.

[153] Object Management Group. *Event Service Specification - Version 1.2*, October 2004. http://www.omg.org/technology/documents/formal/event_service.htm.

[154] Object Management Group. *Notification Service Specification - Version 1.1*, October 2004. http://www.omg.org/technology/documents/formal/notification_service.htm.

[155] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus: an architecture for extensible distributed systems. *SIGOPS Oper. Syst. Rev.*, 27:58–68, 1993.

[156] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *LICS*, 2005.

[157] J. Ouaknine and J. Worrell. On metric temporal logic and faulty turing machines. In *FoSSaCS*, 2006.

[158] J. Ouaknine and J. Worrell. Safety metric temporal logic is fully decidable. *Lecture Notes in Computer Science*, 3920:411–425, 2006.

[159] A. M. Ouksel, O. Jurca, I. Podnar, and K. Aberer. Efficient probabilistic subsumption checking for content-based publish/subscribe systems. *Lecture Notes in Computer Science*, 4290:121–140, 2006.

[160] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30:41–82, 2005.

[161] J. Pereira, F. Fabret, F. Llirbat, R. Preotiuc-Pietro, K. A. Ross, and D. Shasha. Publish/subscribe on the web at extreme speed. In *VLDB*, 2000.

[162] J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In *CooplS*, 2000.

[163] G. Perng, C. Wang, and M. K. Reiter. Providing content-based services in a peer-to-peer environment. In *DEBS*, 2004.

[164] G. P. Picco, D. Balzarotti, and P. Costa. Lights: a lightweight, customizable tuple space supporting context-aware applications. In *SAC*, 2005.

[165] P. R. Pietzuch and J. M. Bacon. Hermes: A distributed event-based middleware architecture. In *ICDCSW*, 2002.

[166] A. J. G. Pinto, J. M. Stochero, and J. F. de Rezende. Aggregation-aware routing on wireless sensor networks. In *PWC*, 2004.

[167] I. Podnar. *Service Architecture for Content Dissemination to Mobile Users*. PhD thesis, University of Zagreb, 2004.

[168] I. Podnar and K. Pripužić. m-newsboard: a news dissemination service for mobile users. In *ConTEL*, 2003.

[169] P. Prabhakar and D. D'Souza. On the expressiveness of mtl with past operators. *Lecture Notes in Computer Science*, 4202:322–336, 2006.

[170] K. Pripužić, H. Belani, and M. Vuković. Early forest fire detection with sensor networks: Sliding window skylines approach. *Lecture Notes in Computer Science*, 5177:725–732, 2010.

[171] K. Pripužić, D. Huljenić, and A. Carić. Vocabulary development for event notification services. In *SoftCOM*, 2004.

[172] K. Pripužić, I. Podnar Žarko, and K. Aberer. Top-k/w publish/subscribe: finding k most relevant publications in sliding time window w. In *DEBS*, 2008.

[173] R. Alur and T.A. Henzinger. Real-Time Logics: Complexity and Expressiveness. *Information and Computation*, 104:35–77, 1990.

[174] R. Alur and T.A. Henzinger. Logics and Models of Real-Time: A Survey. *Lecture Notes in Computer Science*, 600:74–106, 1991.

[175] C. Raiciu, D. S. Rosenblum, and M. Handley. Revisiting content-based publish/subscribe. In *ICDCSW*, 2006.

[176] S. Ramani, K. S. Trivedi, and B. Dasarathy. Reliable messaging using the corba notification service. In *DOA*, 2001.

[177] J.-F. Raskin and P.-Y. Schobbens. State clock logic: A decidable real-time logic. In *HART*, 1997.

[178] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM*, 2001.

[179] W. Rjaibi, K. R. Dittrich, and D. Jaepel. Event matching in symmetric subscription systems. In *CASCON*, 2002.

[180] I. Rose, R. Murty, P. R. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh. Cobra: Content-based filtering and aggregation of blogs and rss feeds. In *NSDI*, 2007.

[181] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[182] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *NGC*, 2001.

[183] E. A. Rundensteiner, L. Ding, T. M. Sutherland, Y. Zhu, B. Pielech, and N. Mehta. Cape: Continuous query engine with heterogeneous-grained adaptivity. In *VLDB*, 2004.

[184] O. K. Sahingoz and N. Erdogan. RUBDES: A rule based distributed event system. *Lecture Notes in Computer Science*, 2869:284–291, 2003.

[185] D. A. Segal, D. K. Gifford, J. M. Lucassen, J. B. Henderson, G. T. Berlin, and D. E. Burmaster. Boston community information system user's manual. Technical report, Massachusetts Institute of Technology, 1987.

[186] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with elvin4. In *AUUG*, 2000.

[187] W. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *AUUG*, 1997.

[188] Z. Shen and S. Tirthapura. Faster event forwarding in a content-based publish-subscribe system through lookup reuseevent. In *NCA*, 2006.

[189] A. P. Sistla. On characterization of safety and liveness properties in temporal logic. In *PODC*, 1985.

[190] A. P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–512, 1994.

[191] T. Sivaharan, G. S. Blair, and G. Coulson. Green: A configurable and re-configurable publish-subscribe middleware for pervasive computing. *Lecture Notes in Computer Science*, 3760:732–749, 2005.

[192] D. Skeen. *Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview*. Vitria, 1998. http://www.vitria.com.

[193] D. Srivastava. Subsumption and indexing in constraint query languages with linear arithmetic constraints. *Annals of Mathematics and Artificial Intelligence*, 8:315–343, 1993.

[194] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.

[195] D. Tam, R. Azimi, and H.-A. Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. *Lecture Notes in Computer Science*, 2944:138–152, 2004.

[196] S. Tarkoma and J. Kangasharju. Optimizing content-based routers: posets and forests. *Distributed Computing*, 19:62–77, 2006.

[197] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *DEBS*, 2003.

[198] S. Voulgaris, E. Rivière, A.-M. Kermarrec, and M. V. Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *IPTPS*, 2006.

[199] T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *ProCoS*, 1994.

[200] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.

[201] M. Wray and R. Hawkes. Distributed virtual environments and vrml: an event-based architecture. *Comput. Netw. ISDN Syst.*, 30:43–51, 1998.

[202] T. W. Yan and H. García-Molina. Index structures for selective dissemination of information under the boolean model. *ACM Trans. Database Syst.*, 19:332–364, 1994.

[203] T. W. Yan and H. Garcia-Molina. The sift information dissemination system. *ACM Trans. Database Syst.*, 24:529–565, 1999.

[204] Y. Zhang. *Computing order statistics over data streams*. PhD thesis, University of New South Wales, 2008.

[205] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and. Technical report, University of California at Berkeley, 2001.

[206] Y. Zhou, K.-L. Tan, and F. Yu. Leveraging distributed publish/subscribe systems for scalable stream query processing. *Lecture Notes in Computer Science*, 4365:20–33, 2007.

[207] Y. Zhu and Y. Hu. Ferry: A p2p-based architecture for content-based publish/subscribe services. *IEEE T. Parall. Distr.*, 18:672–685, 2007.

[208] Y. Zhu and D. Shasha. Statstream: statistical monitoring of thousands of data streams in real time. In *VLDB*, 2002.

[209] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV*, 2001.

# Summary


## Top-k Publish/Subscribe Matching Model Based on Sliding Window

In this thesis we present the top-k/w matching model, a novel publish/subscribe matching model which enables a subscriber to control the number of publications it will receive per subscription within a pre-defined time period. In this model, each subscription defines an arbitrary and time-independent scoring function and parameters $k$ and window-size $w$ such that, at a point in time $t$, parameter $k$ restricts the number of delivered publications to the $k$ best scored publications that are published between $t - w$ and $t$.

We propose novel algorithms for efficient top-k/w processing in environments publishing data objects at high rates. Existing solutions are typically tailored to specific scoring functions, and thus we define a generic data stream processing model that is independent of data representation and scoring functions. Our complexity analysis and extensive comparative evaluation show that our algorithms significantly outperform the competing approaches in both memory consumption and processing efficiency.

For each of the commonly used routing strategies in both centralized and distributed environments we explain how it can be adapted to support the proposed top-k/w matching model. We identify and evaluate routing strategies which are particularly well suited for large-scale top-k/w publish/subscribe systems. Additionally, we present D-ZaLaPS, a distributed top-k/w publish/subscribe system based on a peer-to-peer structured overlay. The experimental evaluation shows that D-ZaLaPS is scalable both for increasing number of peers and subscriptions.

Finally, using the case study with k-nearest neighbor subscriptions, we experimentally evaluate and compare the top-k/w and Boolean matching models. This experimental evaluation shows that, assuming the subscribers are interested in top-k publications in the sliding window, the top-k/w matching model can significantly reduce message overhead in the system when compared to the Boolean matching model, and this, together with built-in support for flexible subscriptions and control of the number of matching publications in the top-k/w model, represents a significant enhancement of publish/subscribe systems.



**Keywords:** data stream processing, publish/subscribe, event-based systems, top-k, sliding-window.

Kratki sažetak

## Model usporedbe ”k-najboljih” u sustavima objavi-pretplati temeljen na klizećem prozoru

Ova disertacija predlaže novi model usporedbe u sustavima objavi-pretplati koji omogućava korisnicima da po svakoj svojoj pretplati kontroliraju broj objava koje žele primiti u odabranom vremenskom intervalu. U ovom modelu usporedbe, pretplate definiraju funkciju za rangiranje objava, parametar $k$ te veličinu vremenskog prozora $w$. U bilo kojem trenutku $t$, parametar $k$ ograničava broj isporučenih objava na $k$ najbolje rangiranih od onih koje su objavljene u periodu između $t$-$w$ i $t$.

U disertaciji se predlaže nekoliko različitih algoritama za obradu ove vrste pretplata u slučaju velikog intenziteta objavljivanja. Iz razloga što u literaturi ne postoje općenita rješenja, već su sva postojeća namijenjena strogo specifičnim funkcijama rangiranja, u disertaciji se predlaže općeniti model obrade ove vrste pretplata koji je u potpunosti neovisan o vrsti podataka i odabranoj funkciji rangiranja. Rezultati eksperimentalne evaluacije i analize složenosti algoritama pokazuju da su predloženi algoritmi znatno učinkovitiji pri odradi podataka te da zauzimaju manje radne memorije od postojećih rješenja.

U disertaciji se predlažu i objašnjavaju preinake uobičajenih strategija usmjeravanja u centraliziranim i raspodijeljenim sustavima objavi-pretplati koje su neophodne za podršku predloženog modela usporedbe. Također se identificiraju strategije koje su posebno pogodne za izvedbu sustava objavi-pretplati koji imaju veliki broj korisnika i podržavaju predloženi model usporedbe. U disertaciji se predstavlja jedan takav sustav temeljen na prekrivajućoj mreži istovrsnih čvorova. Rezultati eksperimentalne evaluacije pokazuju da je ovaj sustav skalabilan pri povećanju broja čvorova i pretplata u sustavu.

Eksperimentalna evaluacija na odabranom studijskom primjeru pokazuje da se broj razmijenjenih poruka u sustavu objavi-pretplati može značajno smanjiti ukoliko se umjesto postojećeg koristi predloženi model usporedbe, što zajedno s fleksibilnošću pretplata i ugrađenom kontrolom intenziteta isporučenih objava u ovom modelu predstavlja značajno poboljšanje sustava objavi-pretplati.

**Ključne riječi**: procesiranje toka podataka, objavi-pretplati, k-najboljih, klizeći prozor.

# Biography

I was born on March 18th, 1980 in Vinkovci, Croatia. After finishing gymnasium (major in natural sciences) in Vinkovci, I started the undergraduate program at the Faculty of Electrical Engineering and Computing, University of Zagreb, in 1998. I received my diploma degree (Dipl.-Ing.) in electrical engineering with a major in telecommunications and informatics from the University of Zagreb, in September 2003. The research topic of my thesis was ”Design and implementation of an application for disseminating multimedia messages in mobile environments”. During academic year 2002-2003 I was a scholarship holder of the State Student Scholarship awarded by the Croatian Ministry of Science, Education and Sports. From October 2003 to July 2004 I worked at the Department of Telecommunications, Faculty of Electrical Engineering and Computing as a research associate, and from July 2004 I work there as a research assistant. In the period between October 2003 and July 2006 I was involved as a researcher on a research project between the Department of Telecommunications and the Research and Development Department of KATE-KOM Company, Zagreb, Croatia. In March 2005 I have started my PhD studies at the Faculty of Electrical Engineering and Computing. As a part of my PhD studies, I spent academic year 2006-2007 at the Distributed Information Systems Laboratory at EPFL (Ecole Polytechnique Fédérale de Lausanne), Switzerland, as a scholarship holder of the Swiss Government scholarship for university, fine arts and music schools for foreign students. My current research interests include data stream processing systems and distributed information systems with special interest in publish/subscribe systems. I have published 6 papers on international conferences and 3 papers in international journals. I am fluent in Croatian and English. I am a member of IEEE.

# Životopis

Rođen sam 18. ožujka 1980. u Vinkovcima. Po završetku srednje škole (prirodoslovno-matematički smjer Gimnazije Matije Antuna Reljkovića u Vinkovcima), 1998. godine upisujem studij na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu, gdje sam i diplomirao u rujnu 2003. godine s naglaskom na znanstveno-istraživačkom radu s temom "Oblikovanje i razvoj aplikacije za isporuku višemedijskih poruka u mobilnom okruženju". Tijekom školske godine 2002./2003. sam primao državnu stipendiju Ministarstva znanosti, obrazovanja i športa Republike Hrvatske. Od listopada 2003. godine sam zaposlen na Zavodu za telekomunikacije Fakulteta elektrotehnike i računarstva na radnom mjestu zavodskog suradnika, a od srpnja 2004. na radnom mjestu znanstvenog novaka kao istraživač na znanstvenim projektima pri Ministarstvu znanosti, obrazovanja i športa. U razdoblju od rujna 2003. do srpnja 2006. sam radio u svojstvu suradnika na istraživačko-razvojnom projektu između Zavoda za telekomunikacije Fakulteta elektrotehnike i računarstva i Odjela za istraživanje i razvoj kompanije KATE-KOM iz Zagreba. Poslijediplomski doktorski studij na Fakultetu elektrotehnike i računarstva upisao sam u ožujku 2005. godine. U sklopu doktorskog studija proveo sam školsku godinu 2006./2007. kao stipendist Švicarske konfederacije u Laboratoriju za distribuirane informacijske sustave na Švicarskom federalnom institutu za tehnologiju u Lausanni. Moja područja istraživanja su sustavi za procesiranje toka podataka te raspodijeljeni informacijski sustavi s naglaskom na sustave objavi-pretplati. Objavio sam 6 znanstvenih radova na međunarodnim konferencijama, 3 rada u međunarodnim časopisima te 1 stručni rad. Govorim hrvatski i engleski jezik. Član sam udruženja IEEE.