CS 237 Final Report (Group 4)
Intelligent subscription service based on the publish-subscribe architecture

Meng-Hsuan Chiang
menghsuc@uci.edu

Chuan-Chun Kuo
chuanchk@uci.edu

Tien-Hsin Lin
tienhsl@uci.edu

Abstract

For the course project of Distributed Systems Middleware, we choose to experiment on several designs of a system that offers intelligence subscription services to Internet users based on the Publish/Subscribe **?** architecture. In this report we give a detailed description of the primary problem we seek to solve, namely the matching problem. We also introduce various approaches we would adopt to tackle this problem. Later on we designed several experiments to test the performance on each approach and discuss about the results. At last we present our findings and include possible improvements we could do in the future.

1 Introduction

Publish/Subscribe (pub/sub) systems are a key technology for information dissemination. In the Publish/Subscribe system, publishers produce information in the form of *events*, which is then forwarded to specific subscribers who register their interest in these events. For example, a user subscription may consist of an interest in a specific type of car which price is lower than a certain amount. A published event may consist of a car with some properties including price. As long as a new published event satisfied the user's requirement, it will be forwarded to the user. The main semantical characterization of pub/sub system is that receivers are not directly targeted from the publisher, but rather they are indirectly addressed according to the content of events.

A distributed pub/sub system for scalable information dissemination can be decomposed in three functional layers: namely the overlay infrastructure, the event routing and the algorithm for matching events against subscriptions. The main architecture is depicted in Figure 1., including three logical layers, namely Overlay Infrastructure, Event Routing, and Matching. We present in the following the functionality associated with each layer:

- **Overlay Infrastructure**: A pub/sub system generally builds upon an application-level overlay network. We choose to implement the most common *Broker Overlay* in our project, which the pub/sub system to be implemented as a set of independent, communicating servers. In this context, each single server is called a *broker*. Clients can access the system through any broker and in general each broker stores only a subset of all the subscriptions in the system.

- **Event Routing**: The core mechanism behind a distributed pub/sub system is event routing. Informally, event routing is the process of delivering an event to all the subscribers that issued a matching subscription before the publication.

- **Matching**: Matching is the process of checking an event against a subscription. Matching is performed by the pub/sub system in order to determine whether dispatching the event to a subscriber or not.

1.1 The Matching Problem

Efficiently matching message topics with interested subscribers is a critical issue in messaging middleware. In topic based, the matching is based on pattern (eg. “forex.*”). A topic is composed of several words, for example, the brand Nike classifies its products for men and women. In the catalog of men, there are several sub-categories too, such as tennis, baseball, and etc. A “*” wildcard character, which matches arbitrarily word, is a bottleneck for message-oriented middleware. Since topics are in hieratical structure, given subscriptions with one or more “*” characters grows the cost of matching in exponentially. As a result, one of our goals is to implement algorithms for accelerating the matching process.

1.2 Synchronization Problem for Multiple Brokers Pub/Sub System

In the multiple brokers’ setting, each broker is responsible for a part of publishers and subscribers. A publisher or subscriber can only connect to a broker. Figure 1 is an example of multiple brokers pub/sub system. There are 3 brokers, each of them has distinct subscribers and publishers connect to it. Since a broker only keeps the subscription information for subscribers related to it. As long as a publisher sending a new message, the corresponding broker forwards the update to others. Consequently, besides the matching problem, an additional issue for multiple brokers system is to synchronize new publish between brokers. The second goal of our project is to simulate the pub/sub system with multiple brokers and build efficient ways to synchronize updates.

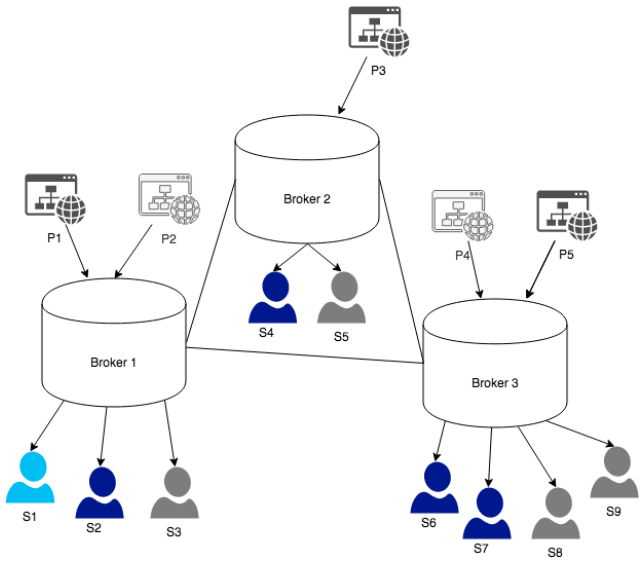


Figure 1: Example of the pub/sub system with multiple brokers

1.3 Following Sections

The following sections are organized as follows: Section 2 is an introduction for algorithms to solve matching and synchronization problems. For the topic matching problem, there are 4 algorithms: naive, inverted bitmap, optimized inverted bitmap, and trie (prefix tree). Section 3 explains some details of our implementation, including class diagrams and program work-flows. In section 4 we analyze the results of our experiments based on the algorithms mentioned above. In section 5 we present the conclusions we obtain from carrying out this project.

2 Algorithms

Subscribers are usually interested in particular events and not in all events. The different ways of specifying the events of interest have led to several subscription schemes.

2.1 On a Single Broker

We have four different algorithms, naive, inverted bitmap, optimized inverted bitmap, and trie, for the matching problem on a single broker.

2.1.1 Naive Method

The naive approach store the mapping between subscriptions and subscribers. As shown in Figure 2, when a broker receives an updated message for a topic, it iterates through the lookup table. If the topic and a subscription are matched, then add following subscribers into the output list. After going through the lookup table, the broker will send update messages to subscribers in the output list.

When there are N subscriptions, and K subscribers, the estimated complexity are: O(1) to insert a subscription into the lookup table, O(M) to search corresponding subscriptions for a new publish and O(N*K) for the space of lookup table.

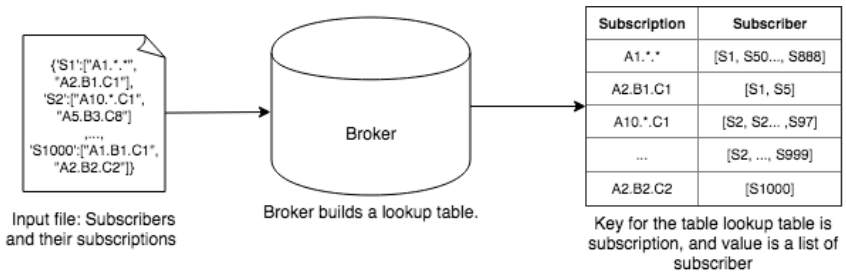


Figure 2: A lookup table in the naive approach

2.1.2 Inverted Bitmap

The inverted bitmap method is based on the observation that lookups are more frequent and search space is finite. The first step is to build a 2-dimensional bitmap consist of finite topics in search space. When subscription comes, we iterate each topic in search space and set the bit (1: match the topic, 0: otherwise) refer to Figure 3 . When a publisher publishes a message, just looks up the corresponding bitmap and sends to subscribers whose set bits is 1.

When there are T topics, and K subscribers, the estimated complexity are: O(T) to insert a subscription into the lookup table, O(1) to search corresponding subscriptions for a new publish and O(T*K) for the space of lookup table.

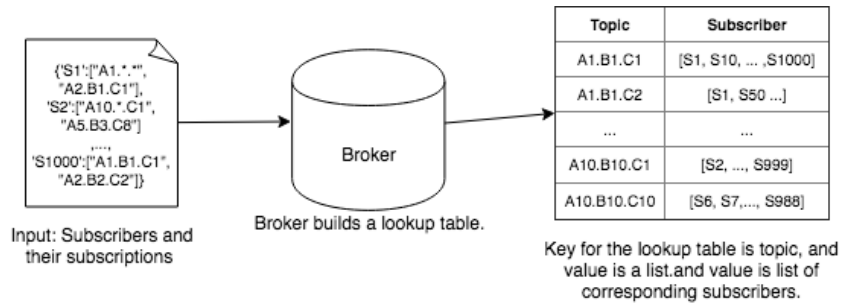


Figure 3: A lookup table in the inverted bitmap approach

2.1.3 Optimized Inverted bitmap

The optimized inverted bitmap works by splitting topics into parts. Each part has a set of a bitmap. When a new message comes, the topic is split it into parts, and each part is matched with the corresponding bitmap, end up resulting bitmaps are joined with AND operation. An improvement from the inverted bitmap is in memory consumption.

When there are W words of each topic, N subscriptions, and K subscribers, the estimated complexity are: O(W) to insert a subscription into the lookup table, O(W) to search corresponding subscriptions for a new publish and O(N*W) for the space of lookup table.

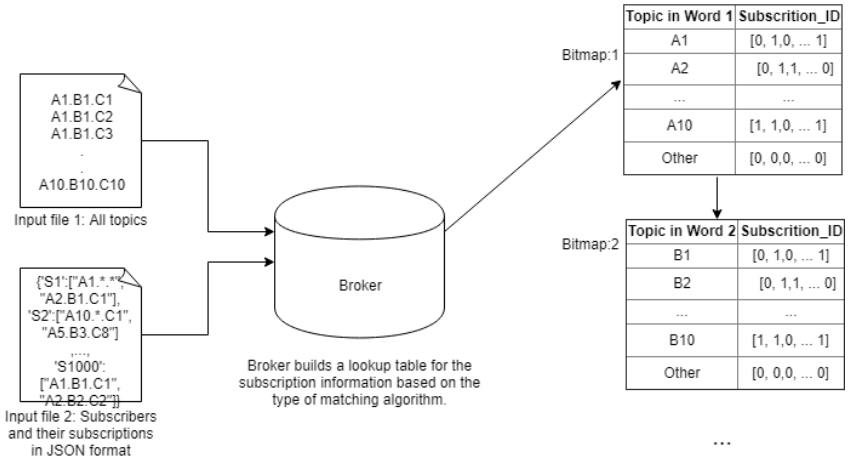


Figure 4: A lookup table in the optimized bitmap approach

2.1.4 Trie

Trie is an extension to the concept of optimized inverted Bitmap. Subscriptions are represented as paths in the tree structure. In Figure 5, metadata is added to the last node of each path to mark a subscription. Matching a subscription with corresponding subscribers is like searching in a trie. There can be wildcard "*" nodes which matches any topic word as well.

When there are W words of each topic, and N subscriptions, the estimated complexity are: O(W) to insert a subscription into the lookup table, O(W) to search corresponding subscriptions for a new publish and O(N*W) for the space of the trie.

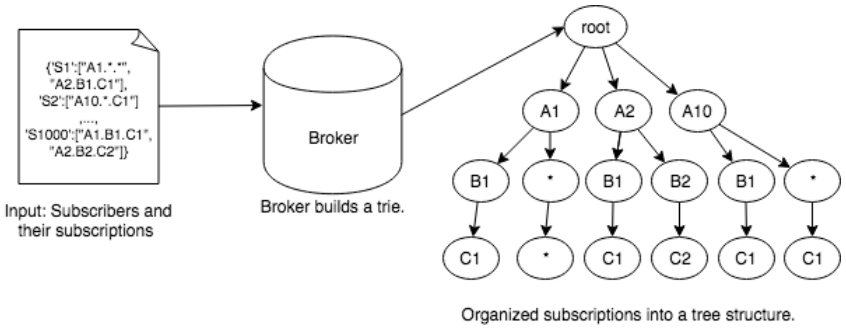


Figure 5: A lookup table in the trie approach

2.2 On Multiple Brokers

To construct a scalable version of our system to carry out studies related to distributed computing, we create a small cluster of 3 brokers. The brokers are connected through the Internet. In order to synchronize messages efficiently we design the multiple broker architecture as a system consisting two servers and a client.

The two server each maintains a connection between the client, while there is no connection required between the two servers. Each server receives messages from the client and sends messages back if necessary. The client sends messages to servers, receives messages from servers and most importantly

forwards messages from one server to the other. In this way the three brokers can exchange messages in a fairly efficient manner.

This idea can be extended to support additional brokers in the network or even a hierarchy of broker networks. After each broker collects all relevant messages for its corresponding group of subscribers, it continues with the topic matching process using any algorithm introduced above and finally forwards message updates to individual subscribers.

3 System Implementation

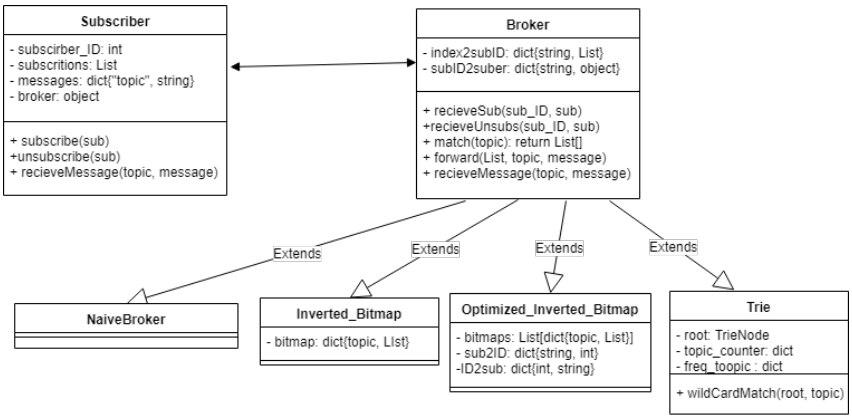


Figure 6: A class diagram for the system

3.1 System Architecture

In Figure 6, we show our system architecture in the form of class diagram. Our system mainly contains three classes: Subscriber, Broker, Publisher. We implement features of each class by scratch in Python. For broker, since there are some common functionality among different algorithms, we make an abstract class which implements the common functions that each type of broker would use. Then for each broker, we simply just need to extend and create their own functions and override parent functions if needed. The subscriber and broker class are associated with each other and hold their reference to make message routing possible.

3.2 Program Flow

In executing phase, we first instantiate the subscribers and assign subscriptions to them. Then we initiate the broker and for each subscriber, we send their subscriptions to the broker and add broker to their reference. The broker would also keep the reference of the subscriptions in its memory. Finally, we instantiate publisher and send the published messages to broker. The broker would then start its matching process to deliver the message to subscribers whose subscription matches the message.

4 Evaluation

4.1 Test on single broker pub/sub system

Our test including two stages, time and space for subscriptions management, and lookup time for publishers to update messages.
First, a broker gets a list of total topics. We assume that a broker knows all of them and no new topic will join during the test. A topic is composed of several words separated by '.' character, the latter word is a subset of the former word. For example, a three words topic Nike.Kids.Training targets on Nike's training products for kids.
Then we evaluate the performance in subscriptions stage (Figure 7). We measure time and space to store the information of users' subscriptions. Each matching algorithm has separated way to hold the

mapping. For example, the naive algorithm stores the mapping between subscriptions and lists of users, and the inverted bitmap stores the mapping between topics and lists of users.
The input is a JSON file includes all users name and their subscriptions. For example, 's1':["A1.*", "A1.B3"], 's2':["A1.B3"] . . . , 's1':["A1.*", "A1.B3"] means subscriber s1 subscribes to topics match to "A1.*" and "A1.B3". A difference between subscription and topic is that the wildcard character only appears in the subscription.
Next, we measure time for publishers to update messages (Figure 8). When the broker receives a message update, it looks up to the subscription table created in the previous stage to find corresponding subscribers. The input in this stage is thousands of randomly generated topics.
For the single broker pub/sub system, we test 4 matching algorithms in various numbers of topics, subscriptions, and update messages.

4.1.1 Test1

Performance for subscriptions management by 4 different matching algorithms. The test files including:

- 100 topics, each of them has 5 words
- 1000 subscribers with totally 2000 subscriptions

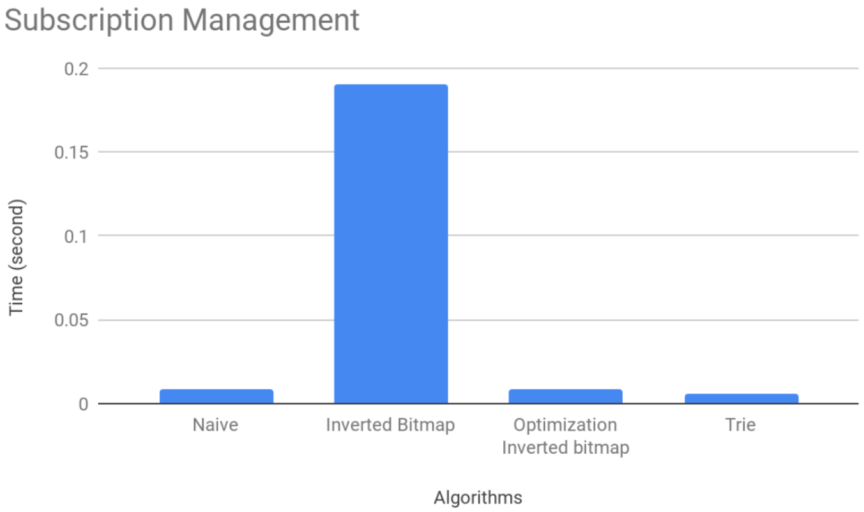


Figure 7: Comparison in subscription time

In the stage of subscription management. The time consumption for the inverted bitmap approach is significantly larger than three other algorithms. A reason is that a subscription is mapped to all possible topics at first. For example, it changes a subscription from "A1.*" into a list of matching topics ["A1.B1", "A1"."B2", "A1.B3" , . . . , "A1.BN"]. As the number of wildcard characters increases, the mapping time grows in exceptional.

4.1.2 Test2

Performance of sending all update messages from publishers to receivers. The test files including:

- 100 topics, each of them has 5 words
- 1000 subscribers with totally 10000 subscriptions
- 10000 random generated update messages

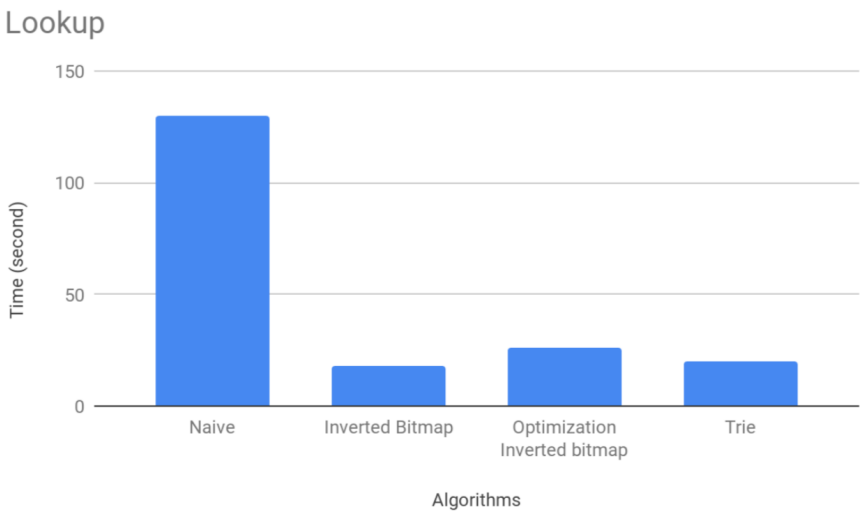


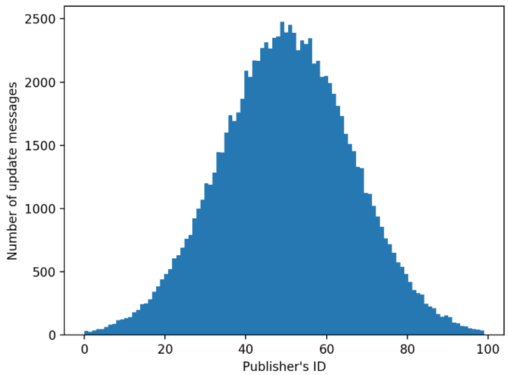
Figure 8: Comparison in lookup time

The Naive approach takes longer time since when a subscription comes, it has to iterative through a whole lookup table. Compares to others, where lookup for a tire simply goes through the tree high, inverted bitmap select from a hash table directly, and the optimized inverted bitmap select from a number of hash tables equal to the number of words in a topic.

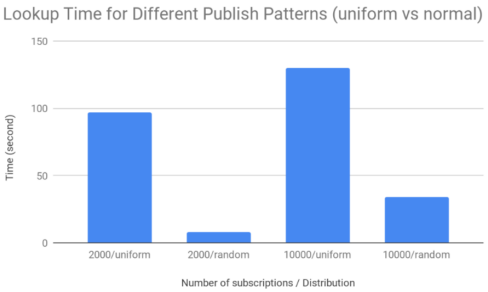
4.1.3 Test3

In the real world, the update frequency varies from different publishers. A few publishers heavily update their messages; while, most publishers update in less frequent. For example, a publisher like weather report renews information constantly; on the other hand, a brand like BMW publishes sales information in relatively low frequency. To simulate the situation, we improve our algorithms by storing the mapping of high frequent publishers and corresponding subscribers into a cache. The frequency of topic is distributed in the normal distribution. (Figure 9a 9b) The test files including:

- 100 topics, each of them has 5 words
- 1000 subscribers with totally 10000 subscriptions
- 10000 update messages in the normal distribution



(a) Normal distribution of update frequency



(b) Results for naive approach

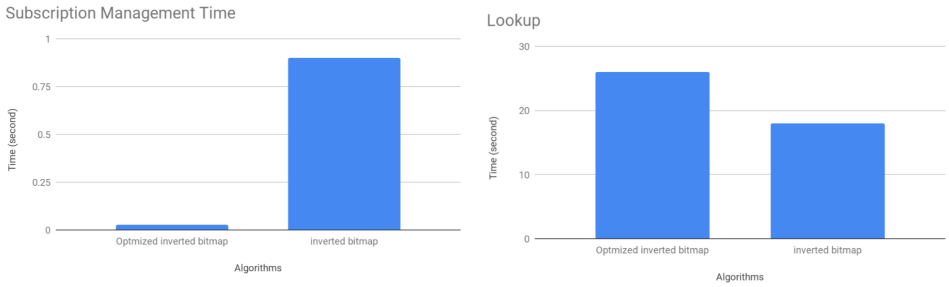
Figure 9

By putting frequently updated publishers into a cache, the improvement is significant. For example, the publisher with ID 50 in Figure 9a has high update frequency, then we add it and its subscribers into a map to save time matching.

4.1.4 Test4

Comparison between *inverted bitmap* and *optimized inverted bitmap* methods. Two approaches are similar; however, the optimized one has multiple bitmaps corresponding to the number of words. As a result, the optimized one performs better in the space and the stage of subscription management; on the other hand, the original one is good in the lookup stage. The test file including :

- 100 topics, with 5 words each
- 1000 subscribers with totally 10000 subscriptions
- 10000 random generated update messages



(a) Comparison of subscription time (b) Comparison of lookup time

Figure 10

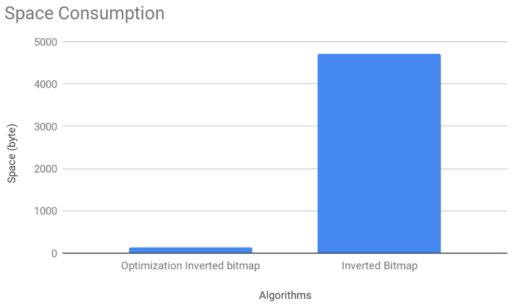


Figure 11: Comparison of space consumption

5 Conclusion

As we've seen, there are several approaches to consider to implement topic matching. We have implemented three solutions by scratch: (1) Naive Hashmap algorithm, (2) Inverted/Optimized Bitmap algorithm, and (3) Trie algorithm. Moreover, after experimenting in a single broker structure, we further adjust the system into multi-broker structure. Last but not least, we perform several experiments to examine the read/write patterns, time complexity, space complexity.
We discovered that the naive hashmap solution is generally a poor choice due to is prohibitively expensive lookup time. Inverted bitmap offer a better solution, and the optimized version is a better choice for scalability, offering a good balance between read and write performance. The trie is indeed the best choice, providing lower latency than other methods and consuming less memory. The only downside is its implementation is more complex than others.

6 References

Roberto Baldoni, Roberto Beraldi, S Tucci Piergiovanni, and Antonino Virgillito. On the modelling of publish/subscribe communication systems.

Concurrency and Computation: Practice and Experience, 17(12):1471–1495, 2005. Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E Strom, and Daniel C Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No. 99CB37003), pp. 262–272. IEEE, 1999.

Kenneth P Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. ACM Transactions on Computer Systems (TOCS), 17(2):41–88, 1999.

Antonio Carzaniga, David S Rosenblum, and Alexander L Wolf. Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems (TOCS), 19(3):332–383, 2001.

Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. ACM computing surveys (CSUR), 35(2):114–131, 2003.

Hojjat Jafarpour, Sharad Mehrotra, and Nalini Venkatasubramanian. A fast and robust content- based publish/subscribe architecture. In 2008 Seventh IEEE International Symposium on Network Computing and Applications, pp. 52–59. IEEE, 2008.

Yi-Min Wang, Lili Qiu, Dimitris Achlioptas, Gautam Das, Paul Larson, and Helen J Wang. Subscrip- tion partitioning and routing in content-based publish/subscribe systems. In 16th International Symposium on DIStributed Computing (DISC'02), pp. 8. Citeseer, 2002.

Lemire, Daniel, et al. "Roaring bitmaps: Implementation of an optimized software library." Software: Practice and Experience 48.4 (2018): 867-895.

Litwin, Witold. "Trie hashing." Proceedings of the 1981 ACM SIGMOD international conference on Management of data. ACM, 1981.

Baldoni, Roberto, et al. "TERA: topic-based event routing for peer-to-peer architectures." Proceedings of the 2007 inaugural international conference on Distributed event-based systems. ACM, 2007.