© 2017 IJSRSET | Volume 3 | Issue 2 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Bloom Filters : A Content Based Prefiltering Technique In Publish/Subscribe system

Poonam B. Linghate, Prof. K. R.Ingole
Department of Computer Science and Engineering, Sipna College Of Engineering and Technology, Maharashtra, India

ABSTRACT

In this paper, we present a content-based publish/subscribe system, called B-PUB/SUB Bloom filter-based pub-sub system. Pub/Sub is a versatile content-based publish/subscribe system. Publish-Subscribe system, distributed entities, called participants, communicate with each other by exchanging messages, often referred to as events. Participants can publish events on the system, or they can subscribe to events of their interest by specifying the type or the content of events they are interested in Publish/Subscribe systems provide a selective dissemination scheme that delivers published content only to the receivers that have specified interest in it. Bloom filters are compact data structures for probabilistic representation of a set in order to support membership queries. This compact representation is the payoff for allowing a small rate of false positives in membership queries.
Keywords: Bloom Filters , Publish-Subscribe, Content-Based.

I. INTRODUCTION

The publish/subscribe(pub/sub) communication paradigms is one of the most used paradigms because of it uses decoupling of publishers from subscribers in terms of synchronization between publisher and subscriber. Publisher published the data in the system and as per subscription, the subscriber received the information. The information published by the publisher are routed to the particular subscriber. Content-based pub/sub is the alternative form that provides the most useful subscription model, where subscription defines restrictions on the message information. Publishers and subscribers do not need to know one another but only exchange data via some pub/sub middleware. Subscribers simply inform the system about what new elements of data they wish to receive by registering a subscription. Conversely, publishers send new events to the system in the form of publications. It is then the responsibility of the pub/sub system to appropriately route each publication towards all subscribers with matching interests.

Burton Bloom introduced Bloom filters in the 1970s, and ever since they have been very popular in database applications Bloom filters are used to test whether an element is a member of a set. Elements can be added to the set, but not removed (though this can be addressed with a "counting" filter); the more elements that are added to the set, the larger the probability of false positives.

Bloom filters are an excellent data structure for succinctly representing a set in order to support membership queries. Bloom filters have been around for ages and are commonly used in Databases and Linguistic applications. For many applications, the probability of a false prediction can be made sufficiently small and the space savings are significant enough that Bloom filters are useful. Bloom filters have a great deal of potential for Distributed protocols where systems need to share information about what data they have available.

II. METHODS AND MATERIAL

1. Background

Hojjat Jafarpour et al.(2012) [1] has proposed a content-based publish/subscribe framework that delivers

matching content to subscribers in their desired format. In our proposed framework, users in addition to specifying their information needs, also specify their profile which includes the information about their receiving context which includes characteristics of the device used to receive the content.

Alessandro Margara et al.(2014) [2] has proposed infrastructure is matching the action of filtering that the core functionality realized by a publish-subscribe each incoming event notification e against the received subscriptions to decide the components interested in e. This is a nontrivial activity, especially for content-based systems, whose subscriptions filter events based on their content. In such cases, the matching component may easily become the bottleneck of the system. For this need parallel hardware are used but unfortunately, moving from a sequential to a parallel architecture is not easy. Often, algorithms have to be redesigned from the ground to maximize the operations that can be performed in parallel and consequently to fully leverage the processing power offered by the platform. This is especially true for GPUs, whose cores can be used simultaneously only to perform data parallel computations.

Christian Esposito et al.(2015)[3] has proposed that publish/subscribe services have encountered considerable success in the building of modern large-scale mission critical systems.

Paolo Bellavista et al.(2014)[4] has proposed a Publish/Subscribe (PUB/SUB) messaging pattern is widely considered as a fundamental way to enable scalable and flexible communication in highly distributed systems. The most significant and recognized advantage of this mode of interaction is the decoupling of communicating parties in space, time, and synchronization.

Tania Banerjee et al.(2015) [5] has proposed Pub/Sub systems are used in diverse applications with varied performance requirements. In some applications,events occur at a much higher rate than the posting/removal of subscriptions while in other applications the subscription rate may be much higher than the event rate and in yet other applications the two rates may be comparable. Optimal performance in each of these scenarios may result from deploying a different data structure for the subscriptions or a different tuning of the same structure. Many commercial applications of pub/sub systems have thousands of attributes and millions of subscriptions. So, scalability in terms of a number of attributes and a number of subscriptions is critical.

2. Literature Review

Hojjat Jafarpour et al.(2012) [1] has proposed DHT based publish-subscribe model which is set of stable nodes as content brokers that are connected through a structured overlay network. Each client connects to one of the brokers and communicates through which it communicates with the system. In DHT-based pub/sub, content space is partitioned among the set of brokers. Each broker maintains subscriptions for its partition of content space and is responsible for matching them against the publications belonging to the same partition. In fact, each broker is the Rendezvous Point (RP) for the publications and subscriptions in its partition. A broker forwards all subscriptions from his own clients to the brokers (RP) responsible for the corresponding content partitions. Similarly, when a broker receives a published content from its client, it forwards the content to the appropriate RP. The content is matched with the list of subscriptions at the RP and the list of brokers with matched subscriptions is created.

Alessandro Margara et al.(2014) [2] has proposed that publish-subscribe content-based matching algorithm designed to run efficiently both on multicore CPUs and CUDA GPUs. At the same time, the analysis identifies the characteristic aspects of multicore and CUDA programming that mostly impact performance.

Christian Esposito et al.(2015)[3] has proposed the logic tree-based schemes which can reduce the management costs of group keys (making them increasingly logarithmic according to the greater group size); however, the communication overhead and rekeying delay still remain high in the case of a large-scale network. In addition, a central key manager has to monitor the status of all group members and keep a fully-connected topology of all the trusted members, implying a high management overhead.

Paolo Bellavista et al.(2014)[4] has proposed a PUB/SUB middleware is a distributed platform that allows its participants to exchange information with each other in the form of events.

Without loss of generality, the middleware assumes an event to be a set of key-value pairs, whose meaning is generally application-dependent. A participant enters information in the system by publishing; it can also express interest in particular events, by means of subscriptions. The middleware delivers events to subscribers according to their subscriptions. In order to subscribe to events, participants select subspaces of the space of all possible events by providing one or more subscription filters, which, in their more general form, are boolean predicates on the event fields. Whenever an event is published, the system will dispatch it to a set of subscribers that have specified a subscription filter that matches the event(i.e., it evaluates to true when applied to the event).

Tania Banerjee et al.(2015) [5] has proposed a PUB/SUB, which is a versatile and scalable, content-based pub/sub system that may be tuned to provide high performance for diverse application environments. PUBSUB is versatile because its architecture supports a variety of predicate types (e.g., ranges, regular expressions, string relations) as well as a heterogeneous collection of data structures for the representation of subscriptions in order to achieve high throughput. The performance of a version of PUBSUB that was tuned for applications in which events occur far more frequently than subscription posting/deletion is compared with the performance of the pub/sub systems BE Tree.

3. Existing Methodologies

In broker less publish-subscribe system approach, publishers, and subscribers communicate with a key server. Credentials are assigned to the key server and in turn, it receives keys which fit the expressed capabilities in the credentials. Subsequently, these keys are used for encryption, decryption, and sign relevant messages in the content-based pub/sub system. The cipher text, are assigned with credentials and the keys are assigned to publisher and subscriber. If there is a match between the identification of the cipher text and the key the particular message gets decrypted. For each authorized credential publisher and subscriber are assigned the private key. The public keys are generated by a string concatenation of a credential, an epoch for key revocation, a symbol (SUB: PUB) which distinguished each publisher from subscribers. There is no need to contact the key server for generating the keys for the communicating system. Similarly, it does not require any middleware for encryption and decryption of event.

In QoS-based services, the involved parties usually perform a quality agreement process to determine the exact service level to be needed at runtime. This process, in the context of PUB/SUB systems, has been often modeled with a publisher offered – subscribed requested (PO-SR) pattern: publishers defines a set of quality properties which they are going to offer for their publications, while subscribers request to the publisher for the desired service level for the delivery of their events. In the view, by concentrating only on participants, this simple agreement model fails at capturing the fundamental role that the middleware has in this process. In fact, in many cases, the middleware distributed components (e.g., the overlay brokers) must have and possibly reserve a nonnegligible amount of computing resources to provision service with guaranteed quality. When a publisher performs a publish action, it can also provide a QoS specification describing the offered QoS. Similarly, advertise actions allow a publisher to declare beforehand the QoS properties it intends to offer for its events, and subscribe actions let a subscriber specify its required quality level.According to this model, for events to match a subscription, it is not sufficient that they satisfy the corresponding subscription filter, but, in addition, the requested and offered quality properties must be compatible, and the middleware must confirm the QoS agreement, possibly allocating the necessary resources.

In Infrastructure-Free content-based Publish/Subscribe, it maps the pub/sub matching problem to a distributed multidimensional indexing problem. In particular, publications and subscriptions are mapped to regions in a multidimensional b space such that the intersection of these region simplifies a match of the corresponding publications and subscriptions. The multidimensional space is partitioned into regions and indexed by a search tree, nodes of which are managed by peers in the network. The indexed regions, as well as subscriptions and publications, are uniquely labeled with keys, which serve to identify peers in the network that manage the corresponding search tree nodes. The keys are designed to allow a search tree node to easily determine the keys of its parent and child nodes, which, again, serve as keys to the underlying DHT to find the relevant peers for these nodes.

Fig. 1 gives the organization of the subscription database used in PUBSUB. This database comprises a collection of level-1 attribute structures A1,,,Am, where m is the number of attributes. It assumes that the allowable attributes have been numbered 1 through m and that the attributes in a subscription are ordered using this numbering of attributes. The attribute structure Ai stores all subscriptions that include a predicate on attribute i but not on any attribute j < i. The attribute i is associated with the structure Ai. With the assumptions on attribute ordering within subscriptions, Ai contains all subscriptions whose first attribute is i. In practice, many of the A is will be empty and only non-empty attribute structures are stored in PUBSUB. The distribution of subscriptions across these buckets is determined by the attribute i predicates in these subscriptions and the data structure D used for keeping the track of the buckets. For uniformity, level-1 attribute structures are associated with a header bucket that is always empty.

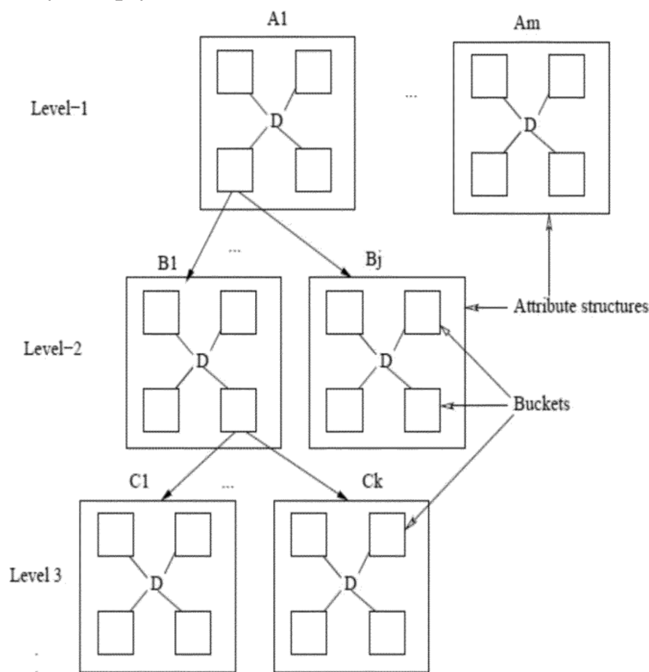

Figure 1: PUB-SUB Organization

III. ANALYSIS AND DISCUSSION

Pub/Sub is a data dissemination model with two entities: 1) Publisher 2) Subscriber

1) Publisher : Publisher is the data producer, publisher provides the data to the related subscribers

2) Subscriber : Subscriber is the consumer, subscribers subscribes to related publisher and receives the data produced by the publisher.

Pub/sub systems are often classified according to the expressiveness of the subscriptions they allow. The model of interest in this paper is content-based filtering using Bloom filters. Access control in the context of pub/sub system means that only authenticated publishers are allowed to disseminate events in the network and only those events are delivered to authorized subscribers.

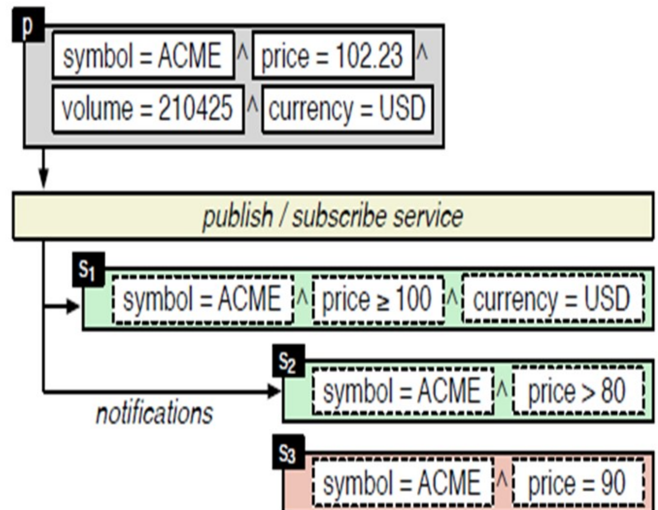

Figure 2 : A content-based pub/sub system.

BLOOM filters are an excellent data structure for succinctly representing a set in order to support membership queries. These Bloom filters encode the values carried by the publications and the equality constraints of the subscriptions. By testing the Bloom filters of subscriptions for inclusion in those of publications, one can efficiently determine the possibility for a message to match a subscription: if the test is negative, the message is guaranteed not to match.

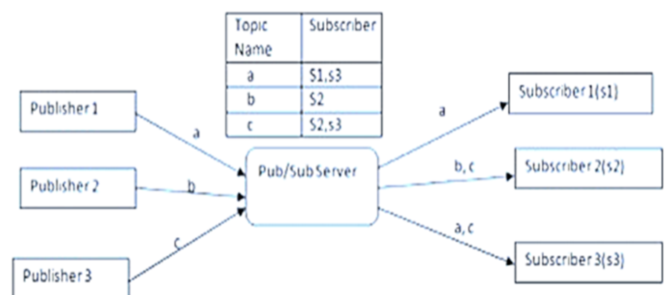

Figure 3: Bloom filter use in pub/sub system.

IV. PROPOSED METHODOLOGY

PUBLISH/SUBSCRIBE, or pub/sub for short is an appealing communication paradigm for building large-scale applications composed of dynamic entities that produce and consume data events according to complex and unpredictable workflows. It offers an indirect, decoupled communication model between application components that act as either publisher of, or subscribers to, data. Publishers and subscribers do not need to know one another but only exchange data via some pub/sub middleware. Subscribers simply inform the system about what new elements of data they wish to receive by registering a subscription. Conversely, publishers send new events to the system in the form of publications. It is then the responsibility of the pub/sub system to appropriately route each publication towards all subscribers with matching interests.

One of the key principles of prefiltering is to quickly identify subscriptions that are known not to match an incoming publication. Bloom Filters are embed in publications and subscriptions, when applicable, and use simple bit-wise operations to discard non-matching subscriptions. The prefiltering uses the data structures shown in Figure 4. We maintain an array (candidates[]) of n + 1 lists of candidate subscriptions, where n is the number of bits in Bloom filters. The I th list for i 2 f0; 1; : : : ; n □ 1g contains subscriptions that have the I th bit of their Bloom filter set. It follows that each subscription may belong to several lists. The last list, called default list, contains subscriptions that have no equality predicate, i.e., an empty Bloom filter.

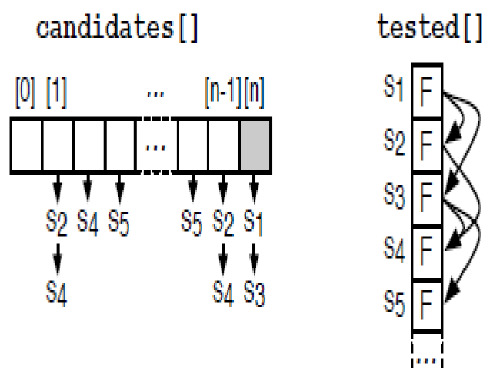

Figure. 4: Data structures used for prefiltering.

A Bloom filter is a simple space-efficient randomized data structure. Bloom filters allow false positives but the space savings often outweigh this drawback when the probability of an error is made sufficiently low. Bloom filters are probabilistic data structures that allow for efficient testing of whether or not an item belongs to a set. A Bloom filter is essentially a bit array. When adding an item, one or several hash functions h1; : : : ; hk are used to identify bit(s) of the Bloom filter that must be set to 1. To test whether an item belongs to the set, it is hashed again and, if all corresponding bits are set, it is a likely member the set; otherwise, it is guaranteed not to be. Therefore, Bloom filters can yield false positives but no false negatives. The accuracy of the Bloom filter can be tuned by properly choosing its size and the number of hash functions.

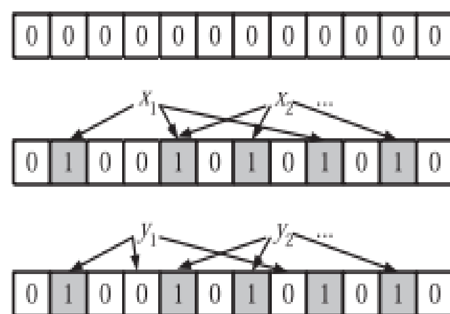

Figure 5. An example of a Bloom filter.

The filter begins as an array of all 0s. Each item in the set xi is hashed k times, with each hash yielding a bit location; these bits are set to 1.

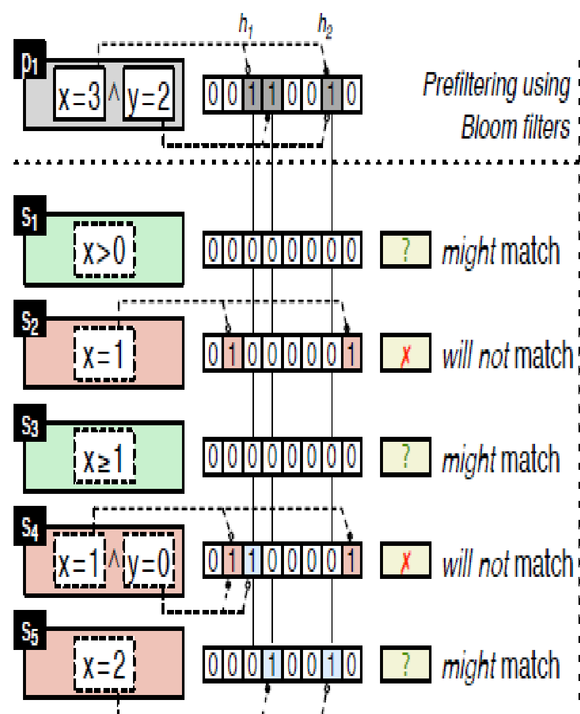

Figure. 6: Information used for prefiltering: Bloom filters embedded with subscriptions and publications.

Bloom filters[6] is embed in publications and subscriptions, when applicable, and use simple bit-wise operations to discard non-matching subscriptions. When injecting a publication p in the system, besides encrypting it, we additionally hash the values of all its fields and insert them in a non-encrypted Bloom filter B(p). On the other side, every subscription s also embeds a Bloom filter B(s) containing the hashed values of its predicates with equality constraints.

For instance, given a predicate x=2, the value 2 will be added to the Bloom filter. Figure 6 shows the information that can be used for prefiltering for a publication p1. We assume for the illustration filters of 8 bits and two hash functions h1; h2. Upon matching, if B(s) * B(p),1 we know that s does not match p, irrespectively of the other (non-equality) predicates, and we can discard it. This is the case for s2 where two bits set in B(s2) are not set in B(p1). During prefiltering, we only need to traverse the candidate lists associated with the bits set in the publication's Bloom filter, as well as the default list. All other subscriptions are discarded. The number of lists to traverse is therefore function of the number of fields (i.e., attributes) of the publication, which is expected to be much smaller than the size of the Bloom filters.

V. CONCLUSION

By adding Bloom filters that encode publication values and subscription equality constraints, it can discard a large fraction of subscriptions before reaching the costly encrypted filtering operation. Evaluations confirmed that our mechanisms reduce the number of such costly operations required to filter an incoming publication by approximately one order of magnitude. Prefiltering technique makes use of Bloom filters. This data structure is also used in other work relating to confidentiality or security purposes in distributed systems.

VI. FUTURE SCOPE

From Observation, the scope and planned to be studied in future work, Compressed Bloom Filters can use. By using compression can improve Bloom filter performance, in the sense that it can achieve a smaller false positive probability as a function of the compressed size over a Bloom filter that does not use compression.

VII. REFERENCES

[1]. Hojjat Jafarpour, Bijit Hore, Sharad Mehrotra, and Nalini Venkatasubramanian "CCD: A Distributed Publish/Subscribe Framework for Rich Content Formats " , Ieee Transactions On Parallel And Distributed Systems, Vol. 23, NO. 5,PP.844-852 May 2012.

[2]. Alessandro Margara and Gianpaolo Cugola "High-Performance Publish-Subscribe Matching Using Parallel Hardware", Ieee Transactions On Parallel And Distributed Systems, Vol. 25, No. 1, Pp. 126-135,January 2014.

[3]. Christian Esposito and Mario Ciampi "On Security in Publish/Subscribe Services: A Survey", IEEE Communication Surveys & Tutorials, Vol. 17, No. 2, Pp.962-997, May 2015.

[4]. Paolo Bellavista and Andrea Reale "Quality of Service inWide Scale Publish–Subscribe Systems" IEEE communications surveys tutorials, vol. 16, no. 3, PP.1591-1616 third quarter 2014.

[5]. Tania Banerjee and Sartaj Sahni" PUBSUB: An Efficient Publish/Subscribe System" IEEE Transactions On Computers, Vol. 64, NO. 4,PP.1119-1132, April 2015.

[6]. B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Comm. of the ACM, vol. 13, no. 7, 1970.