Dynamic Load Balancing for Cluster-based Publish/Subscribe System

Hojjat Jafarpour, Sharad Mehrotra and Nalini Venkatasubramanian
Department of Computer Science
University of California, Irvine
{hjafarpo, sharad, nalini}@ics.uci.edu

Abstract

In this paper we present load balancing techniques for Cluster-based pub/sub framework that include both static and dynamic load balancing. The static load balancing is done through a multi-cluster architecture for broker overlay network which is based on subscription distribution knowledge in the event space. The dynamic load balancing, on the other hand, is achieved through exploiting multiple inter-cluster and intra-cluster connections to dynamically distribute publication and subscription forwarding load among brokers during run time. Our experimental results show that the proposed load balancing techniques effectively prevent overloaded brokers without having significant effect on content dissemination efficiency.

1. Introduction

Content-based publish/subscribe (pub/sub) is a content distribution paradigm where a message is routed based on its content rather than specific destination address attached to it [1, 2, 3]. Subscribers specify their interest in certain events and will be notified afterward if a published event matches their interest. By decoupling communication parties, pub/sub service provides anonymity and asynchrony which makes it an attractive communication infrastructure for many applications. For scalability reasons, a large-scale, content-based pub/sub systems is often implemented as a distributed service network where a set of dedicated broker servers form an overlay network. Clients connect to one of these brokers and publish or subscribe through that broker. When a broker receives a subscription from one of its clients, it acts on behalf of the client and forwards the subscription in the broker overlay. Similarly, when a broker receives an event from its client it forwards the event through the broker overlay to the brokers that have matching subscriptions. These brokers then deliver the event to their interested clients.

Different architectures for organizing the dedicated broker servers in an overlay network have been proposed in the literature. Most of the existing pub/sub systems either organize broker servers in an acyclic graph structure [7, 9] or a structured overlay network based on distributed hash table [12, 11]. We refer to the former approach as the *Tree-based* pub/sub and latter one as *DHT-based* pub/sub. Cluster-based pub/sub is a new architecture for content-based pub/sub that is not only resilient to broker failures, but also provides fast content dissemination and load balancing among brokers [4]. It organizes event brokers in clusters where each broker is connected to all brokers in the cluster it belongs to and at least one broker in every other cluster. Subscription propagation is limited to clusters resulting in reduced subscription dissemination and storage load. Event dissemination is done in two phases. An event first is disseminated among clusters. Then, it is matched with subscriptions in each cluster and is delivered to the brokers with matching subscriptions. Cluster-based pub/sub also provides fault tolerance in case of broker failures through multiple connections between clusters and subscription replication in clusters. It speeds up event dissemination by reducing the number of brokers(hops) an event travels to reach to its subscribers and parallelizing content matching operation. It scales to high publication and subscription load by distributing publication and subscription load among all brokers.

In this paper, we propose load balancing strategies to uniformly distribute the publication dissemination load among brokers and prevent overloaded brokers which may result in broker failures. We present a static load balancing strategy based on a multi-cluster architecture which reduces broker load by exploiting the subscription distribution knowledge. In multi-cluster strategy, the popular sections of the event space use the overlay with smaller clusters while unpopular sections of the event space use the overlay with larger clusters. We also present dynamic load balancing strategies which distribute publication dissemination load by offloading the load from overloaded brokers to underloaded broker in run time. Our experimental results show that the load balancing in Cluster-based pub/sub is signif-

icantly better than the existing Tree-based and DHT-based systems. Our results also shows that even if the dissemination load is very skewed, the proposed load balancing techniques efficiently distribute the load among brokers and prevent overloaded brokers.

The rest of the paper is organized as follows. In the next section we present a brief overview of the Cluster-based pub/sub system. Section 3 represents our load balancing strategies including the multi-cluster architecture and dynamic load balancing algorithms. Our experimental results are presented in Section 4 followed by review of related work in Section 5. Section 6 concludes the paper.

2 Cluster-based Publish/Subscribe System

In this section we present a brief overview of cluster-based pub/sub framework. For an in depth description of the cluster-based pub/sub we refer the reader to [4]. We assume the system consists of a set of dedicated broker servers, B=$\{B_1,..., B_n\}$ where brokers communicate through reliable TCP links and have unique identification numbers. The cluster-based pub/sub partitions brokers into clusters where each broker belongs to one cluster. Brokers in a cluster maintain connections with one another and can directly communicate. Besides the brokers in its cluster, each broker also maintains connections with at least one broker in every other cluster which forms a *ring*. A ring consists of a set of brokers, one from each cluster. A broker B_i uses *ClusterBrokerList*$_i$ and *RingBrokerList* to keep the list of brokers in its cluster and ring respectively. Since each broker can be in only one cluster, it has only one *ClusterBrokerList*$_i$. On the other hand, a broker can have one or more *RingBrokerList* meaning that it can be part of multiple rings. Finally, B_i stores subscriptions and subscribers from its clients in *LocalSubscriptionList*$_i$ and all subscriptions and their subscriber brokers in the same cluster in *ClusterSubscriptionList*$_i$. Detailed information about the number of clusters and cluster size and forming the cluster network is provided in [4]. Figure 2 illustrates a sample system with nine brokers forming three clusters with three rings.

2.1 Subscription Dissemination

A client sends its subscriptions which contain its ranges of interest to its corresponding broker. When a broker B_i receives a subscription from its client, it first looks into *LocalSubscriptionList*$_i$ to see if the subscription is covered by previous subscriptions it has received from its clients. If the subscription is covered, the broker just adds it to the *LocalSubscriptionList*$_i$. Otherwise, after adding the subscription to its *LocalSubscriptionList*, the broker disseminates the subscription among all other brokers in the same

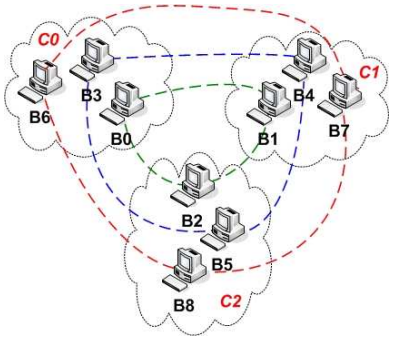

Figure 1. Sample system with 9 brokers partitioned into 3 clusters.

cluster. Each broker B_j in the cluster after receiving the subscription, first looks into its *ClusterSubscriptionList* and removes all subscriptions from B_i that are covered by the new subscription. Then it adds the new subscription from B_i to its *ClusterSubscriptionList*. The motivation in using covering relation among subscriptions is to reduce the number of stored subscriptions in brokers by removing covered subscriptions from same subscriber [6].

2.2 Event Dissemination Algorithm

The event dissemination is done in two phases. The first phase is *Ring* dissemination where a published event is broadcast among all clusters through the publisher broker's ring. In the second phase, which is *Cluster* dissemination phase, the event is matched to subscriptions in each cluster and is delivered to the brokers with matching subscription. The formal representation of the dissemination algorithm is depicted in Algorithm 1.

Algorithm 1 Event Dissemination Algorithm

1: $B_{pub} \leftarrow$ The publisher broker
2: PubRing $\leftarrow$ The publisher broker's ring
3: **Ring Phase:**
4: **for all** $B_i \in$ PubRing **do**
5: B_{pub} sends the content to B_i
6: **end for**
7: **Cluster Phase:**
8: **for all** $B_i \in$ PubRing **do**
9: B_i matches the content with subscriptions in its cluster
10: B_i sends the content to the matched brokers in its cluster
11: **end for**

3 Load Balancing in Cluster-based Pub/Sub

In this section we define different components of broker load in Cluster-based pub/sub and present our static and dynamic load balancing techniques for Cluster-based pub/sub framework. But before discussing the details of our proposed load balancing techniques, it is worh to note that

even without load balancing capability, the Cluster-based pub/sub outperforms tree-based and DHT-based pub/sub frameworks in reducing and also distributing the dissemination load among brokers. We present the experiment results validating this claim in Section 4.

3.1 Broker Load Components

Since the computation load resulting from content matching heavily depends on content format, in this paper we focus on broker network load. The broker network load results from two sources. Part of broker load results from publications and subscriptions of its clients which we call *client load*. The other part of the load, which we call *forwarding load*, results from forwarding publications and subscriptions from other brokers in rings or clusters. Since clients connect to closest broker, we assume client load cannot be transferred to other brokers. Therefore, in this section we focus on balancing forwarding load among brokers. The forwarding load of a broker consists of three components. The *ring publish load* resulted from publication broadcast in ring(s) that the broker belongs to, the *cluster publish load* resulted from forwarding publications to brokers with matching subscriptions in the same cluster and finally the *cluster subscription load* which results from receiving subscriptions from other brokers in the same cluster. Each of the forwarding load components is composed of send and receive loads where the send part is due to forwarding publications or subscriptions and the receive part results from receiving publications and subscriptions.

3.2 Load Balancing Through Multi-cluster Architecture

The load balancing through multi-cluster architecture is a static load balancing technique which is based on the relationship between the number of clusters and the content dissemination and subscription maintenance load. In our initial clustering approach, we assumed we have just one clustering. However, if some statistical information about subscriptions and publications such as their distribution is available, the broker clustering process can be done more intelligently to reduce subscription maintenance load and speed up content dissemination. In most of the pub/sub systems, user interest distribution follows Zipf or uniform models[5]. In uniform distribution, all parts of event space are equal in probability of having subscription and publication. However, in Zipf distribution of subscriptions, some parts of the event space are more popular and there are more subscriptions for events falling in these parts. On the other hand, there are less popular parts of event space where there are fewer subscriptions for events falling in these parts. Therefore, if we have different clustering for each of these parts in event space, we can achieve better distribution of subscription maintenance load. This means that the number of clusters and cluster sizes will be different for each of event space parts based on their popularity and brokers use the corresponding clusters based on the event space part a publication or subscription falls into. We propose a clustering rule for forming clusters based on the distribution of user interest on event space. In this method the content space is partitioned based on the distribution of subscriptions and for each partition one clustering is used.

Clustering Rule: The number of clusters for a content space partition is directly related to the popularity of that partition. The more subscription in the partition, the more number of clusters for that partition's clustering.

Assuming the number of publications and subscriptions for matching ratio r are p and s respectively, we can achieve the total network traffic for m clusters in the system using equation (1).

$$Overal\ Network\ Traffic(r) = p[(m-1) + r(n-m)] + s(\frac{n}{m} - 1)$$

Based on this equation to achieve minimum network traffic, the number of clusters, m, must be $\sqrt{\frac{s}{p(1-r)}n}$. Therefore, if a part of event space is more popular, there will be more subscriptions in the system for that and also the matching ratio will be larger. This results in bigger value for m and consequently, larger number of clusters. On the other hand, if a part of event space is less popular, it means that there is fewer subscriptions and also the matching ratio is smaller. Thus for achieving less network traffic, the number of clusters must be small.

Let us explain the rationale behind the multi-cluster architecture through the following example. Assume we partition content space into three partitions $\{P_0, P_1, P_2\}$ where P_0 is the most popular partition with large number of subscriptions from majority of brokers and P_2 is the least popular partition with small number of subscriptions from a small fraction of brokers. By assigning more number of clusters for P_0 the size of clusters become smaller. Since each broker just maintains subscriptions from brokers in its own cluster, the smaller cluster size results in smaller number of subscriptions to maintain for each cluster which results in less subscription storage and faster content matching. Also since most of brokers have subscribed to events in this partition, ring dissemination phase does not have considerable adversary effect. This is because of higher probability of having subscriber in all clusters which justifies broadcasting publications to all of clusters in ring. On the other hand, by having fewer clusters for P_2, despite cluster sizes become larger and each broker should maintain subscriptions from larger number of brokers, the subscrip-

tion maintenance load does not increase significantly since fewer brokers subscribe for content in partition P_2. Fewer number of clusters, on the other hand, results in faster dissemination because in ring dissemination phase event is broadcast among fewer brokers which is faster and in cluster dissemination phase just interested brokers receive content.

3.3 Dynamic Load Balancing

In this section we propose strategies for load balancing in a Cluster-based pub/sub system that can be employed dynamically during content distribution process. Our dynamic load balancing strategies focus on balancing all components on forwarding load in the broker overlay network. Depending on factors such as broker's processing power, message queue size and broker's bandwidth each broker can handle certain amount of messages in a time unit. We assume the maximum messaging rate that a broker can handle is R. We say a broker is overloaded when the rate of messages it receives, processes or sends is higher than a certain threshold. Here we assume the threshold is $0.9R$. A broker calls load balancing process when its load reaches the threshold. The load balancing module then tries to offload the extra load to other brokers in the system. This is done by offloading one or more load components in the broker. We also assume that when a broker is overloaded, it does not accept incoming ring content. The load components that can be offloaded in a broker include *ring publish load*, *cluster publish load* and *cluster subscription load*.

Balancing ring publish load: When a broker is overloaded, the first step it takes to reduce the load is by calling ring publish load balancing module. In this case, if the broker is publishing content, the ring publish load balancing module reduces the broker's load by offloading the extra ring dissemination load to other brokers in its cluster that have underloaded rings. Brokers in the same cluster can exchange their current load by piggybacking on disseminated content. This information can help an overloaded broker to find the appropriate underloaded broker for offloading. The overloaded broker also can find the proper broker for load offloading by querying the brokers in the same cluster to discover if they can take over some ring dissemination load. After finding the underloaded broker, a portion of the publishing load is sent to this broker to be disseminated. The receiving broker treats these incoming content as the content that is published by itself and initiates ring dissemination phase.

It is also possible that some of the brokers in the ring become overloaded and do not accept the incoming ring publications. In this case, the publishing broker finds an underloaded broker in its cluster as described above and forwards the publications to the clusters of the overloaded brokers through the underloaded broker. Note that the ring dissemination load for a publication is split with the selected underloaded broker. The ring publish load balancing is depicted in the first phase of the publication dissemination algorithm in Algorithm 2.

Algorithm 2 Event Dissemination Algorithm

```
1:  B_pub ← The publisher broker
2:  PubRing ← The publisher broker's ring
3:  Ring Phase:
4:  if B_pub's ring is overloaded then
5:      B_pub finds broker B_k in its cluster with an underloaded ring
6:      B_pub forwards the publication to B_k
7:      for all B_i ∈ B_k's ring do
8:          if B_i is overloaded then
9:              B_k finds B_k̄ in B_k's cluster and deliver content
10:             to B_i's cluster through B_k̄'s ring
11:         else
12:             B_k sends the content to B_i
13:         end if
14:     end for
15: else
16:     for all B_i ∈ PubRing do
17:         if B_i is overloaded then
18:             B_k finds B_k̄ in B_k's cluster and deliver content
19:             to B_i's cluster through B_k̄'s ring
20:         else
21:             B_pub sends the content to B_i
22:         end if
23:     end for
24: end if
25: Cluster Phase
26: for all B_i ∈ PubRing do
27:     if B_i is overloaded then
28:         B_i finds B_j ∈ B_i's cluster that is underloaded
29:         B_i forwards the publication to B_j
30:         B_j matches the content with subscriptions in its cluster
31:         B_j sends the content to the matched brokers in its cluster
32:     else
33:         B_i matches the content with subscriptions in its cluster
34:         B_i sends the content to the matched brokers in its cluster
35:     end if
36: end for
```

Balancing cluster publish load: Cluster publish load results from matching publication with cluster subscriptions and forwarding it to brokers with matching subscriptions in the cluster. When cluster publish load for a broker B_i changes its state into overloaded, the broker starts looking for another broker in the cluster that can accept the extra load. B_i queries brokers in its cluster by sending a load balancing request to each of them. When a broker B_j receives the load balancing request and can handle part of B_i's extra load, it replies with an ACCEPT message along with the amount of load it can accommodate. When B_i receives ACCEPT message from B_j, it forwards portion of its cluster publish load to this broker and B_j treats these publications as its own cluster publications and matches and disseminates them among brokers. The cluster publish load balancing is depicted in the second phase of the publication dissemination algorithm in Algorithm 2.

Balancing cluster subscription load: Compared to publication dissemination load, we assume that the subscription generation rate in system is considerably smaller. If a broker's cluster subscription load makes it overloaded,

the broker notifies subscriber broker(s). The subscriber brokers then stop subscription dissemination for a randomly selected period of time and then resume dissemination of subscription in the cluster. During this random period, the overloaded broker forwards the publications that it has received from its ring(s) to randomly selected brokers in its cluster and these brokers perform content matching and dissemination among matching brokers in the cluster. When the overloaded broker can accept the new subscriptions after the time period pasts, it notifies the subscribing brokers to send their new subscriptions to this broker again.

4 Evaluation

In this section we present our experimental methodology and simulation results for evaluating our proposed load balancing techniques in Cluster-based pub/sub system.

System setup: To evaluate our algorithms we developed a message level, event-based simulator. Since the focus of this paper is content dissemination among brokers, we performed our simulations only for the broker overlay.

Data model: One of the main challenges in evaluating pub/sub systems is lack of real-world application data. However, previous work shows that in most applications events and subscriptions follow Zipf or uniform distributions [5]. For comprehensiveness, we did our experiments with both of these distributions. We use *Matching Ratio* as our main parameter [13]. Matching ratio is the fraction of the brokers that have matching subscriptions for an event. Using wide variety of matching ratios in our simulations, the results can be interpreted for both Zipf and uniform distributions. High and low matching ratio implies Zipf distribution where some events are very popular and have many subscribers while other events are very selective and a small fraction of brokers have subscribers for these events. Average matching ratio implies uniform distribution where the probability of subscription is almost equal for all events. Since we do not concentrate on content matching techniques in this paper, we ignore the overhead of content matching in our simulations and use a simple presentation of content.

The simulations were done with a broker network consisting of 100 brokers. For multi-cluster architecture the brokers partitioned brokers into 5, 10 and 20 clusters. For dynamic load balancing we partitioned the brokers into 10 cluster each with 10 brokers. Due to space limitation, we only present two sets of our experimental results in this paper. We first compare load distribution in Cluster-based pub/sub with representative implementations of Tree-based and DHT-based pub/sub system. We then provide our experimental results for evaluating the effect of the proposed load balancing techniques in the Cluster-based pub/sub system.

Custer-based pub/sub vs. Tree-based and DHT-based pub/sub: The first set of our experimental results represent the comparison of multi-cluster and single cluster Cluster-based pub/sub with two other common pub/sub broker overlay architectures, Tree-based and DHT-based. The simulation is based on the number of events that is handled by a broker in one time unit and the publications and subscriptions are uniformly distributed among brokers. Figure 2 presents the dissemination load distribution for three different matching ratios. As it is depicted, the Tree-based pub/sub performs the worst in case of load distribution. In all the represented matching ratios, there are several brokers with very high amount of load, while there are other brokers with very small amount of load. This can be justified based on the tree structure of the broker overlay where any publication from one side of the tree that has matching subscription in the other side of the tree must pass through the brokers in the middle. These brokers may end up with processing almost all the publications which results in high dissemination load on them. On the other hand, brokers in the edge of the overlay do not participate in publication forwarding very often which results in very small load. The DHT-based system performs slightly better that the Tree-based one, however, in all cases our Cluster-based systems, multi-cluster and single-cluster, disseminate the load more uniformly among brokers and avoid highly overloaded brokers. This results from breaking up the forwarding load among different rings which not only results in smaller amount of dissemination load but also limits the forwarding load of each publication in its publisher's ring.

Another notable fact in the results is the benefit of using multi-cluster architecture. As it can be seen, when the number of clusters is small, the load distribution also is more uniform which is because of smaller dissemination load resulting from smaller number of clusters. This clearly shows the benefit of using multi-cluster architecture where the subscription distribution in the event space is available.

Dynamic load balancing: The second set of results we present here is the evaluation of the proposed dynamic load balancing algorithms. In order to represent the significance of the dynamic load balancing techniques, we compare the Cluster-based pub/sub with and without load balancing. Similar to the previous experiments subscription distribution among brokers is uniform, however, the publication distribution follows Zipf distribution where a small number of brokers publish majority of publications. The experiments are based on the 100 publication in one time unit and we set the broker load threshold to 150 messages per time unit. The broker load in our experiments consists of input and output message traffics that a broker handles. Figure 3 depicts the load distribution in Cluster-based architecture with and without load balancing for three different matching ratios, 25%, 50% and 75%. The results clrearly

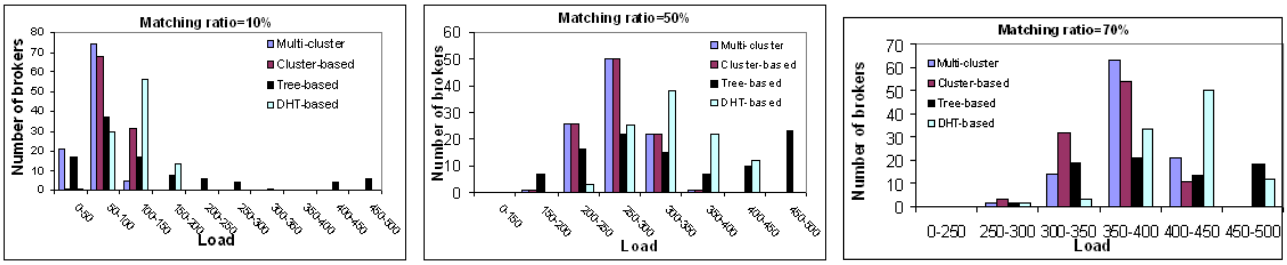

Figure 2. Event dissemination load distribution for 10%, 50% and 70% matching ratio

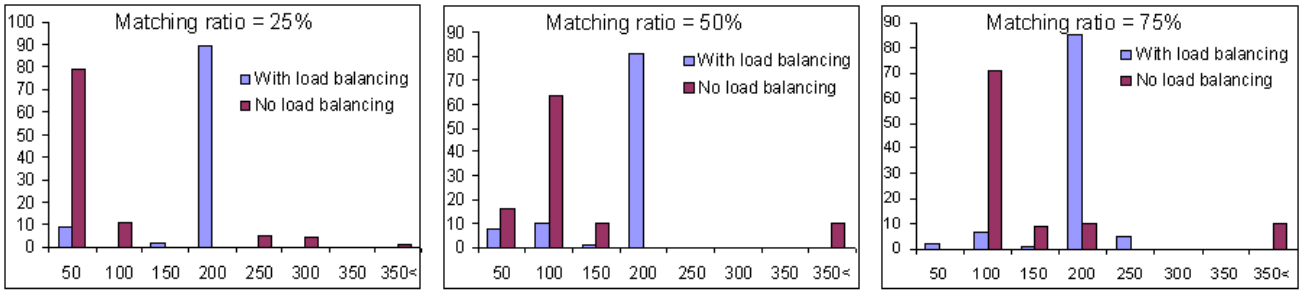

Figure 3. Event dissemination load distribution for 25%, 50% and 75% matching ratio. The X and Y axes represent the load range and the number of brokers, respectively.

show that when the publication distribution among brokers follows Zipf distribution, the Cluster-based pub/sub without load balancing results in uneven distribution of dissemination load. This is because of concentration of disseminiation load on a small portion of rings in the overlay which results higher dissemination load on the brokers on these nodes while the other brokers in the other rings remain underoaded. On the other hand, in Cluster-based pub/sub with dynamic load balancing, the rings with higher load off load portion of the load on the other ring with smaller load which results in more uniform distribution of load among brokers. Consider the graph for 25% matching ratio in figure 3. As it can be seen, in pub/sub with dynamic load balancing, almost 90% or brokers have a load in [200,250) and there is no broker with higher load. However, in the same situation, if the load balancing techniques are not employed, the load distribution is very skewed and more than 75% of brokers have a load in [50,100) while around 10% of brokers have a load higher than 250 and in fact there is one broker with load more than 350. The same results is achieved for other matching ratios which justifies the benefit of using our dynamic load balancing techniques in efficiently distributing the load among brokers and preventing overloaded brokers as much as possible.

5 Related Work

Most of content-based pub/sub systems that use Tree-based content routing including [7, 3], do not provide any load balancing mechanism. Cheung and Jacobsen in [10] proposed a dynamic load balancing scheme for content-based pub/sub system where brokers are organized in a hierarchical (Tree-based) structure. Brokers with more than one neighboring broker are referred to as cluster-head brokers, while brokers with only one neighbor are referred to as edge brokers. Publishers connect to cluster-head brokers, while subscribers are connected to edge brokers. The proposed scheme allows for two levels of load balancing: local-level where edge brokers within the same cluster load balance with each other; and global-level where edge brokers from two different clusters load balance with each other. The main drawback of this scheme is may migrate subscriptions from one broker to another which not only is an extra load on brokers, but also does not preserve subscription locality. Unlike Cluster-based pub/sub architecture which provides transparent fault tolerance, it is not clear how the hierarchical (Tree-based) architecture overcomes the broker failures.

As a DHT-based pub/sub framework, Meghdoot proposed an interesting load balancing scheme [12]. In Meghdoot the content space is partitioned among brokers and each broker is responsible for one of the partitions. The subscriptions are routed to and stored in the corresponding bro-

kers for their partitions. Each publication is also routed to the broker responsible for the partition that the publication falls in. This broker is referred to as Rendezvous Point. After receiving the publication, the Rendezvous Point matches the publication with the subscriptions and routes the publication to the brokers with matching subscriptions. The overloaded brokers can offload part of their load by dividing their partition into two section and transmit the responsibility of one section along with the corresponding subscriptions to another underloaded broker. However, this may require retransmission of subscriptions which results in extra load.

Shuffle is another DHT-based pub/sub framework that provides load balancing [15]. Unlike Meghdoot, in Shuffle subscriptions are stored in all brokers in the path from the subscribing broker to the Rendezvous Point. An overloaded broker exploits the replication of subscriptions and designate an underloaded broker in its children list in the dissemination tree to take the responsibility of content matching and forwarding to the subscribers in its subtree. This scheme relies on the fact that the path for forwarding publication is the reverse path of subscription dissemination that may not be accurate if there are failures in the broker network.

6 Conclusions

We propose load balancing strategies to prevent overloaded brokers in a Cluster-based pub/sub system by distributing the publication dissemination load among brokers. Our proposed techniques include a static load balancing using a multi-cluster architecture to reduces broker load by exploiting the subscription distribution knowledge. We also proposed dynamic load balancing strategies that can offload the load from overloaded brokers to underloaded broker in run time. We validate the effectiveness of the proposed strategies through evtensive simulations. Our experimental results show that not only the load balancing in Cluster-based pub/sub is significantly better than the existing Tree-based and DHT-based systems but the proposed load balancing techniques also efficiently distribute the load among brokers and prevent overloaded brokers even if the dissemination load is very skewed.

References

[1] Patrick Th. Eugster , Pascal A. Felber , Rachid Guerraoui , Anne-Marie Kermarrec, *The many faces of publish/subscribe*, ACM Computing Surveys, v.35 n.2, p.114-131, June 2003.

[2] R. Baldoni, R. Beraldi, S. Tucci Piergiovanni, A. Virgillito, *On the Modelling of Publish/Subscribe Communication Systems*, Concurrency and Computation: Practice and Experiences, Volume 7, issue 12, pages 1471-1495, John Wiley and Sons.

[3] A. Carzaniga, M. Rutherford and A. Wolf, *A Routing Scheme for Content-Based Networking.*, IEEE INFOCOM 2004.

[4] H. Jafarpour, S. Mehrotra and N. Venkatasubramanian, *A Fast and Robust Content-based Publish/Subscribe Architecture.*, IEEE NCA 2008.

[5] A. Riabov, Z. Liu, J. Wolf, P. Yu and L.Zhang , *Clustering Algorithms for Content-based Publication-Subscription Systems.*, IEEE ICDCS 2002.

[6] H. Jafarpour, B. Hore, S. Mehrotra and N. Venkatasubramanian, *Subscription Subsumption Evaluation for Content-based Publish/Subscribe Systems.*, Middleware 2008.

[7] A. Carzaniga, D.S. Rosenblum and A. Wolf, *Design and Evaluation of a Wide-Area Event Notification Service.*, ACM Trans. on Computer Systems, (19)3, Aug 2001.

[8] A. Carzaniga and A. L. Wolf, *Forwarding in a Content-Based Network.*, ACM SIGCOMM 2003.

[9] G. Li, S. Hou, H. A. Jacobsen, *A Unified Approach to Routing, Covering and Merging in Publish/Subscribe Systems Based on Modified Binary Decision Diagrams.*, ICDCS 2005.

[10] A. K. Y. Cheung and H. A. Jacobsen, *Dynamic Load Balancing in Distributed Content-Based Publish/Subscribe.*, Middleware 2006.

[11] R. Baldoni, C. Marchetti, A. Virgillito, R. Vitenberg, *Content-Based Publish-Subscribe over Structured Overlay Networks.*, ICDCS 2005.

[12] A. Gupta, O. Sahin, D. Agrawal, A. El Abbadi, *Meghdoot: Content-Based Publish/Subscribe over P2P Networks.*, Middleware 2004.

[13] F. Cao, J. Pal Singh, *MEDYM: Match-Early with Dynamic Multicast for Content-Based Publish-Subscribe Networks.*, Middleware 2005.

[14] M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, S. Mehrotra, *CREW: A Gossip-based Flash-Dissemination System.*, IEEE ICDCS 2006.

[15] H. Zhang, S. Ganguly, S. Bhatnagar, R. Izmailov and A. Sharma, *Optimal Load Balancing in Publish/Subscribe Broker Networks*, IEEE ICC 2008.