A Fast and Robust Content-based Publish/Subscribe Architecture

Hojjat Jafarpour, Sharad Mehrotra and Nalini Venkatasubramanian
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Email: {hjafarpo,sharad,nalini}@ics.uci.edu

Abstract

We present cluster-based publish/subscribe, a novel architecture that is not only resilient to event broker failures but also provides load balancing and fast event dissemination service. Our proposed approach achieves fault tolerance by organizing event brokers in clusters. Multiple inter-cluster links provide continuous availability of dissemination service in presence of broker failure without requiring subscription retransmission or reconstruction of broker overlay. Furthermore, the proposed architecture provides a fast event dissemination infrastructure that significantly reduces subscription and publication dissemination traffic and load on event brokers. Our experimental results show that even in the presence of 10% failure rate in broker network, event dissemination is not interrupted and dissemination speed and load are not affected significantly.

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]. Subscribers specify their interest in certain events and will be notified afterward if a published event matches their interest. 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.

Most of the existing pub/sub systems including [2,4] use content-based routing algorithm introduced in [1, 2]. These systems construct a spanning tree on the pub/sub overlay network to forward subscriptions and publications. This allows avoiding diffusion of content in parts of the pub/sub network where there are no subscribers and prevents multiple deliveries of events. The other reason for using spanning tree is to exploit subscription aggregation to reduce subscription maintenance overhead. We refer to this approach as *tree-based* pub/sub. Recently, some pub/sub systems that use structured overlay networks for organizing broker servers have been proposed [5, 6]. These systems partition content space and map each partition to a broker using a hash function. Subscriptions and publications are routed to their corresponding brokers, *Rendezvous Point (RP)*, using DHT-based routing techniques such as Chord [7], Tapestry [19] or CAN [8]. RP then matches event with stored subscriptions and routes the event towards the brokers that their subscriptions matched using the DHT-based routing.

Despite the rich functionalities that existing systems provide, most of these techniques assume a stable broker overlay network and do not deal with broker failure. The main problem that arises in case of broker failure is loss of stored subscriptions in the failed broker which results in inaccurate content routing or even interruption in content dissemination process. Some tree-based pub/sub systems deal with broker failure by establishing redundant paths in the broker overlay [10]. However, this results in sending every subscription and event over multiple routes, resulting in increased bandwidth consumption. Some DHT-based approaches such as [6, 12] propose fault tolerance by having back up for brokers but if both broker and its back up crash, subscriptions belonging to their partition are lost and content routing will be interrupted.

In this paper, we propose 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. Our proposed architecture, called *Cluster-based pub/sub*, 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. Our proposed approach 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 subscribers and parallelizing content matching operation.

The rest of the paper is organized as follows. In the next section we present the system design, subscription and publication propagation algorithms. Then, we present the algorithms for dealing with changes in the broker overlay network resulting from broker join and leave and failures. We present our experimental results in Section 4 followed by review of related work in Section 5. Section 6 concludes the paper and provides some directions for future work.

2 Cluster-based Publish/Subscribe System

2.1 System Model and Notations

We assume the architecture of pub/sub system consists of a set of dedicated brokers, B=$\{B_1,..., B_n\}$. Brokers communicate through reliable TCP links and have unique identification numbers. We also assume a fail-stop failure model for broker overlay network that means once a broker fails, it remains in that state and this can be detected by other brokers [15]. Since we assume links between brokers are reliable, we focus on broker failures and ignore other failures such as message lost.

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 from 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*$_i$. Figure 1(a) illustrates a sample system with nine brokers forming three clusters with three rings.

2.2 System Initialization

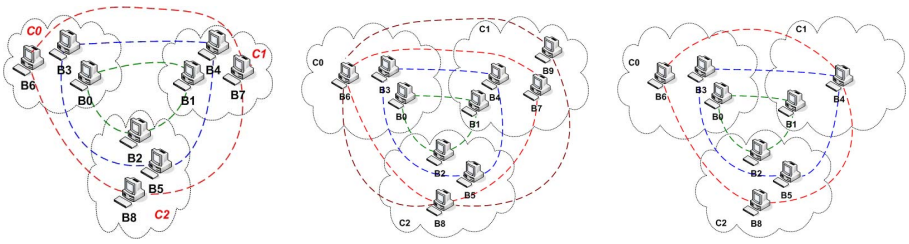
Before the system starts its work, clusters and rings should be formed. There are two main questions that should be answered in clustering brokers. 1) How many clusters should be in the system? 2) How many brokers should be in each cluster? The answer for these questions directly affects the performance of the proposed pub/sub framework in both message forwarding load experienced by a single broker and overall network traffic for dissemination of subscriptions and publications. Assuming subscriptions and publications are distributed uniformly among brokers, we want to form clusters and rings in such a way that subscription and publication dissemination load is uniformly distributed among brokers. We argue that this is achieved if the size of clusters are the same.

Consider the subscription load on a broker is the number of subscriptions that it receives and should store and ring publication load on a broker is the number of events that it receives in the ring dissemination phase. It is straightforward to show that if the size of clusters differ significantly, the distribution of subscription and publication loads among brokers will not be uniform. Thus, in order to distribute both of these loads uniformly, at the initialization time our systems minimizes the difference of cluster sizes and generate almost equal clusters.

Assuming that the cluster sizes should be equal, the next step is to find the number of clusters in the system. As mentioned, the goal is to achieve better performance through reducing overall network traffic resulted from dissemination of subscriptions and publications. Therefore, we first compute the overall network traffic in the system. Assume that there are n brokers in the system and we clustered them into m clusters with equal sizes. Also consider that the probability for a broker to have subscription matching with a publication is represented by $r(0 \leq r \leq 1)$, which we refer to as matching ratio. The total network traffic is the sum of publication forwarding traffic and subscription forwarding traffic. The publication forwarding traffic resulted from one publication in the system is computed as $(m-1)+r(n-m)$. The first part of this expression is the ring dissemination traffic where the publication is disseminated to $m-1$ clusters and results in $(m-1)$ messages. In this phase m brokers have the published message. Now considering the matching ratio is r, $r(n-m)$ brokers out of the remaining $(n-m)$ brokers have matching subscriptions and must receive the message. This results in an extra $r(n-m)$ messages. Therefore, if we have p publication with matching ratio r in the system, the overall publication traffic for matching ratio r becomes $p[(m-1)+r(n-m)]$. Assuming that publications are uniformly distibuted for matching ratios, the overall publication traffic for all matching ratios is computed as follows.

$Publication\ traffic = \int_0^1 [p(m-1)+p(n-m)r]dr$
$= p(m-1)+\frac{p}{2}(n-m)$

On the other hand, since subscriptions are only disseminated in a cluster, the overall subscription dissemination traffic in the system can be computed as follows where s is total number of subscription in the system.



(a) Sample clustering. (b) B9 joins the system. (c) B7 leaves the system.

Figure 1. Sample broker network after join and leave of a broker.

$Subscription\ traffic = s(\frac{n}{m} - 1)$

Using the computed values we can compute the overall network traffic resulted from publication and subscription dissemination.

$Overall\ traffic = p(m-1) + \frac{p}{2}(n-m) + s(\frac{n}{m} - 1)$

Assuming that the overall network traffic is a function of m, the number of clusters in the system, we can achieve the value for m that results in the minimum overall network traffic as follows.

$Number\ of\ Clusters = m = \sqrt{\frac{2s}{p}n}$

This equation shows that the initial number of clusters in the system has a direct relation with the number of subscriptions and an inverse relation with the number of publications. By assuming the amount of publications is twice the amount of subscriptions our system uses $\sqrt{n}$ as the initial number of clusters.

After finalizing the initial number of clusters, the clusters and rings are formed using the initial brokers. One way to do this is that system administrator sets up the *ClusterBrokerList* and *RingBrokerList* for each broker and establishes the connections. The other technique to construct the system structure is through an incremental process where brokers join the system one by one. In this case, the first m brokers form clusters and the first ring in the system. Then other brokers are added to the system by joining to a cluster and forming corresponding ring. The details of join process are described in the next section.

2.3 Subscription and Unsubscription

When a broker B_i receives a subscription from its client, it first adds it to the *LocalSubscriptionList*$_i$ and looks into *LocalSubscriptionList*$_i$ to see if the subscription is covered by previous subscriptions it has received from its clients. If the subscription is not covered, the broker disseminates the subscription among all other brokers in the same cluster. Each broker B_j in the cluster after receiving the new subscription from B_i adds it to its *ClusterSubscriptionList*.

When a client sends an unsubscription request to its broker B_i, the broker first looks into its *LocalSubscriptionList*$_i$. If the subscription was covered, the broker just removes it from *LocalSubscriptionList*$_i$. Otherwise, it finds all subscriptions that have been covered by this subscription in *LocalSubscriptionList*$_i$ and puts them into a list called *UncoveredSubs*. Then broker B_i sends this list along with the request for unsubscription to all other brokers in the same cluster. Any broker B_j in the cluster that receives this information, first removes the subscription from its *ClusterSubscriptionList* and then adds the subscriptions in *UncoveredSubs* to its *ClusterSubscriptionList*.

2.4 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. The most straightforward way to disseminate event among all brokers in the ring is that B_i sends it to all brokers in its *RingBrokerList*. However, if event size is large, faster broadcast dissemination techniques such as the one presented in [16] can be used to speed up and scale ring dissemination. 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. Any efficient content matching technique can be used in this phase [3]. The formal representation of the dissemination algorithm is depicted in figure 2.

$B_{pub} \leftarrow$ The publisher broker
PubRing $\leftarrow$ The publisher broker's ring
Ring Phase
1) For all $B_i \in$ PubRing
   B_{pub} sends the content to B_i
Cluster Phase
1) For all $B_i \in$ PubRing
   B_i matches the content with subscriptions in its cluster
   B_i sends the content to the matched brokers in its cluster

---

Figure 2. Event Dissemination Algorithm

3 Changes in Broker Network

In this section we describe how our proposed system deals with changes in broker overlay network in pub/sub system. Changes in broker overlay network can be caused

by adding new brokers, removing existing brokers and failure of some brokers.

3.1 Broker Join

When a new broker wants to join to the system, it should become part of a cluster and a ring in the broker overlay network. Assume broker B_{new} wants to join the system and knows at least one of the brokers in the system. It first contacts a broker in the system and receives its *RingBrokerList* which consists of one broker in each cluster. The new broker then requests brokers in the *RingBrokerList* to send it the size of their clusters. Based on this information, the new broker finds the cluster with the fewest number of brokers and joins to this cluster. This minimizes the difference between cluster sizes in the system which is one of the objectives in forming the clusters. After selecting the cluster to join, B_{new} requests the *ClusterBrokerList* from the broker in the selected cluster. It also receives the *ClusterSubscriptionList* for the selected cluster. To join the cluster, B_{new} sends a join request to each broker in the cluster. Each broker after receiving the join request from the new broker, adds B_{new} to its *ClusterBrokerList* and sends the number of its rings to the new broker. After receiving the number of rings from these brokers, if there is a broker with more than one ring, the new broker receives the *RingBrokerList* of one of the rings from the broker and substitutes that broker with itself in the *RingBrokerList*. This becomes new broker's *RingBrokerList*. Now the new broker sends a request to the brokers in its *RingBrokerList* and asks them to replace the previous broker with the new on in their *RingBrokerList*. Otherwise, if all the brokers have only one ring, the new broker chooses one broker randomly and receives its *RingBrokerList*. Similar to the previous case the new broker sends join requests to all brokers in its *RingBrokerList* but in this case it asks them to add a new ring which contains the new broker. Figure 1(b) depicts the broker overlay network in figure 1(a) after adding broker B_9.

3.2 Broker Leave

When a broker B_i wants to leave, it first sends a leave request to all brokers in its cluster. Brokers in the cluster respond to the request by sending the number of rings that they belong to and also by removing subscriptions from B_i from their *ClusterSubscriptionList*. When B_i receives the response from the brokers, it selects the broker with the minimum number of rings, B_j, and sends its *RingBrokerList* to B_j. Now B_j sends a message to all brokers in the *RingBrokerList* that it received from B_i and informs them about B_i's leave. If all brokers in the *RingBrokerList* have another ring containing a broker in B_j's cluster, they send a message to B_j and inform it that the ring is deleted and all of these brokers delete this ring. Otherwise, all brokers after receiving this message from B_j, replace B_i with B_j in their corresponding *RingBrokerList*. Then, B_j sends a message to B_i and informs it that it can leave the system and B_i simply leaves the system. Figure 1(c) represents the structure of broker overlay in figure 1(a) after broker B_7 leaves the system.

3.3 Broker Failure

As mentioned in Section 2, one of the main goals of our approach is to provide fault tolerance in such a way that despite failure of some brokers, event dissemination service remains available and works correctly. Here we describe our approach for detecting failures, masking them in order to provide continuous service availability and eventually recovering from them.

Failure Detection: As mentioned in Section 2, we assume brokers follow a fail-stop failure model. We use two strategies to detect failures. The first strategy is based on heart beat messages that brokers in clusters and rings exchange. In this case, each broker periodically sends probe messages to the brokers in the same cluster and ring. If a broker stops receiving heart beat messages from another one for a certain period of time, it assumes that the broker has failed. The other way of failure detection is done during event dissemination. When a broker wants to send a message to another one, if the TCP connection cannot be established after a certain period of time, the sender broker assumes that the other broker has failed and uses failure masking algorithm. It is possible that link between two brokers fails. In this case, we assume the underlying network will eventually fix the failed route and brokers can communicate with each other before connection establishment time expires. If the sender broker cannot establish the connection after this time, it concludes that the receiver broker has failed.

Masking Failures: If a broker fails while content dissemination is in progress it is desired that the failure be masked for the dissemination process so it can progress without being interrupted. In this case there is no need for restructuring broker overlay network and retransmitting subscriptions. To achieve this goal, we propose a modified version of content dissemination algorithm that tries to deliver content to all clusters despite failure of some brokers. The algorithm is presented in figure 3.

We explain the algorithm with respect to the sample in figure 1(a). Assume broker B_4 publishes an event and broker B_3 has failed. Now in the ring phase of dissemination, B_4 cannot deliver the event to B_3. Therefore, B_4 contacts another broker, for instance B_1, in its cluster and asks it to deliver the event to cluster C_0 which is the cluster that B_3 belongs to. If B_1 also cannot deliver the event to the cluster, it asks another broker, here B_7, to send the event to cluster C_0. This process continues until the event gets delivered to the cluster unless all the links between the two clusters have failed. Lets assume B_0 is alive. As a result,

B_1 sends the event to B_0 and B_0 matches the event and disseminates it among brokers with matching subscriptions in Cluster C_0. The adapted dissemination algorithm can disseminate events as long as there is one link between any two clusters. This condition holds until the failure of k brokers in the two clusters where the cluster size is k. Therefore, as long as the number of broker failures is less than k, cluster connectivity exists and masking algorithm can disseminate event despite failures.

B_{pub} ← The publisher broker
PubRing ← The publisher broker's ring
Ring Phase
  For all $B_i \in$ PubRing
    if(B_i has failed)
      Find $B_f \in B_{pub}$.ClusterBrokerList and send the content
      to B_i.Cluster through B_f
    else
      B_{pub} sends the content to B_i
Cluster Phase
  For all $B_i \in$ PubRing
    if(B_i has failed) ignore it.
    else
      B_i matches the content with subscriptions in its cluster
      B_i sends the content to the matched brokers in its cluster

---

Figure 3. Failure masking event dissemination algorithm

Note that it is possible some of events do not get delivered to all subscribers because of failures. For instance if a broker after receiving an event in ring dissemination phase fails before or during dissemination of the event among matched subscribers in its cluster, the subscribers in the cluster do not receive the event. An acknowledgement based extension of the algorithm can solve this problem to the cost of adding acknowledgement load.

Failure Recovery: By masking failures, our approach can provide continues pub/sub service. However, when the number of failures increases, it increases content dissemination time and load. It also results in higher load on some brokers that are in the clusters of failed brokers. Therefore, after detection of broker failure, our proposed system repairs the broker overlay network structure by reconstructing the affected rings and replacing the failed broker with another one in its cluster for the rings that have been affected by failure. The recovery process is similar to broker leave process except that here another broker initializes recovery process.

4 Evaluation

In this section we present our experimental methodology and simulation results for our cluster-based approach. We compare our proposed architecture with two major existing architecture, tree-based pub/sub and DHT-based pub/sub.

4.1 Experimental Methodology

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 [**?**]. For comprehensiveness, we did our experiments with both of these distributions. We use *Matching Ratio* as our main parameter [9]. 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.

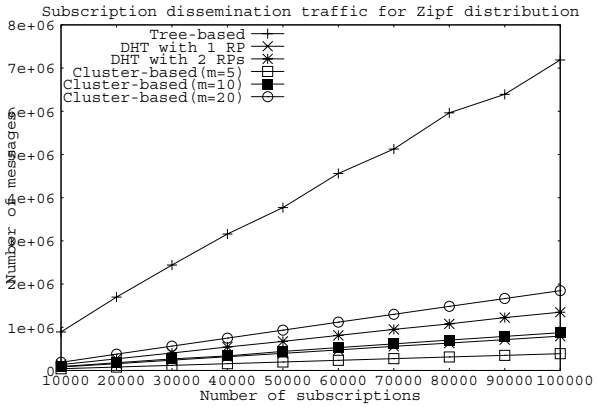


Figure 4. Subscription dissemination traffic.

Simulation setting: We performed our simulations on top of J-Sim real-time network simulator [17]. The network topology is generated by GT-ITM random graph generator using the transit-stub model [18]. There are 20 transit domains with an average of 5 routers in each. There are 3 stub domains attached to each transit router in average and each stub domain has an average of 8 routers. All together there are 2500 routers and 10660 links. 100 brokers are randomly attached to routers by links with latency between 10 to 90 ms. Subscriptions and events are uniformly distributed among brokers and measurements are averaged for 500 publications. Events are small java objects with maximum size of 1KB. The simulations were done for three different settings with 5, 10, 20 clusters. In the overlay setting with 10 clusters we considered two faulty cases where 5% and 10% of brokers have failed. Failures are distributed uniformly. The connection establishment time out that is used for detecting failures is set to 30 seconds which means if a broker cannot establish a TCP connection to another one in 30 seconds it assumes the other broker has failed. We also sim-

ulated a tree-based pub/sub system where brokers are connected through a spanning tree and a DHT-based pub/sub using Tapestry [19] routing scheme that utilizes dynamic multicast technique proposed in [5] to achieve higher speed and less network traffic.

To evaluate subscription dissemination load we used a five dimensional event space and considered subscription covering relation between subscriptions. We generated subscriptions using Zipf distribution. Dimension domain is [0,1000] and each subscription range size is at most half of the dimension domain.

Subscription dissemination traffic. Dissemination of subscriptions among brokers in a pub/sub system imposes significant traffic to the broker overlay network. Traditional tree-based pub/sub systems broadcast subscriptions in the broker overlay network. However, to reduce subscription dissemination load they use subscription covering to prevent further dissemination of covered subscriptions. In DHT-based pub/sub depending on the event space partitioning a subscription may be stored in more than one broker. Also in case of having back up for RPs, a subscription should be replicated on back up brokers too. On the other hand, in cluster-based pub/sub, subscription dissemination is limited to clusters which significantly reduces subscription propagation load compared to tree-based approach.

Figure 4 depicts the subscription dissemination traffic in tree-based, DHT-based and cluster-based pub/sub with different number of clusters for Zipf distribution. As it can be seen, subscription dissemination traffic in tree-based pub/sub is significantly higher than DHT-based and cluster-based pub/sub. The main reason, as mentioned, is that subscriptions are broadcast in tree-based pub/sub and even though the covering relation among subscriptions reduces subscription dissemination load, this reduction just prevents small fraction of subscriptions from being broadcast. On the other hand, in cluster-based pub/sub, although the use of subscription covering is limited to subscriptions from the same broker, since the subscription dissemination is limited to clusters, it generates considerably less amount of traffic. Also as it is shown, subscription dissemination traffic for cluster-based system with fewer number of cluster is higher that the traffic in a system with more clusters. The reason is that the fewer clusters means that the cluster sizes are bigger. This results in dissemination of subscription among more brokers that generates higher amount of traffic. Note that the resulted traffic from DHT-based pub/sub is also considerably small and is almost equal to the Cluster-based approach. This is because subscriptions are forwarded only to the small subset of brokers and use of dynamic multicast to reduce the number of sent messages [5]. However, since subscriptions belonging to a specific event space partition are stored in one or at most two brokers if there is back up for brokers, this results in more vulnerability for loosing stored subscriptions in DHT-based pub/sub approach.

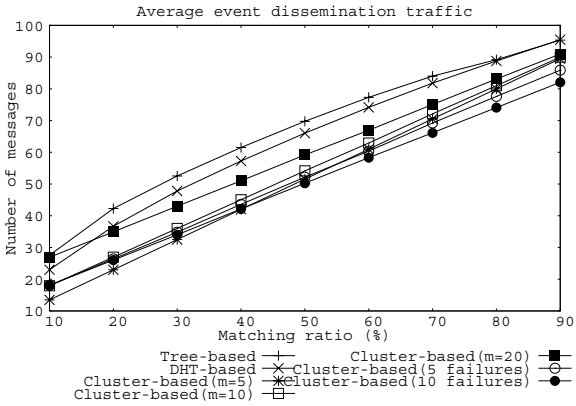


Figure 6. Average event dissemination traffic.

Event dissemination latency. In most of pub/sub applications event dissemination time is a critical factor and it is desired to disseminate publications among subscribers as fast as possible. To evaluate the dissemination time in our approach we measured average dissemination time which is the average time it takes from publishing an event till brokers with matching subscription receive it. We also measured Total dissemination time which is the amount of time takes a published event be delivered to all brokers with matching subscriptions. Figure 5 plots the average and total dissemination time. As it can be seen, our proposed approach in all settings outperforms traditional tree-based and DHT-based approaches. Also the difference between results for different number of clusters is not significant. The main reason is that the number of brokers an event should travel to reach to its subscriber broker is the same (two brokers, one in the ring dissemination and the other in the cluster dissemination) despite the difference in the number of clusters. Also dissemination time stays steady for all matching ratios. This is because of uniform distribution of brokers in tree-based and DHT-based approaches and constant number of hops (two) that an event passes in our cluster-based approach. The other important fact depicted in the graphs is that despite 5% or 10% broker failures in our approach, still average dissemination time is considerably less than the time for tree-based approach without failures. Also the average dissemination speed in cluster-based pub/sub is less than DHT-based approach without any failure.

Event dissemination traffic. We assume event dissemination traffic as the average number of transmitted messages between brokers during event propagation process . Figure 4.1 plots the average event dissemination traffic. As it can be seen, there is a direct relation between the popularity of an event and the amount of traffic is generated for its dissemination. For less popular events where matching ratio is small the dissemination traffic also is small. By increas-

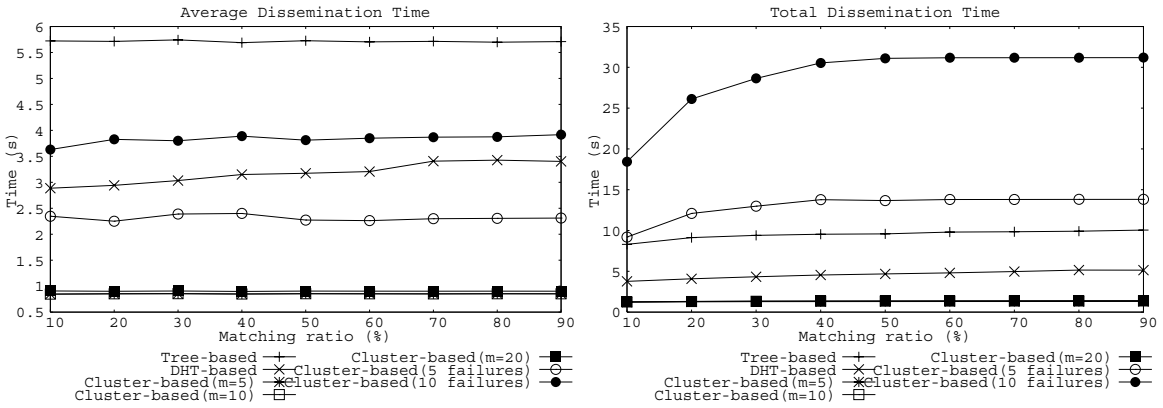


Figure 5. Event dissemination times.

ing the number of subscribers, event should be forwarded to more brokers which results in higher traffic. The cluster-based approach, specially with fewer clusters, performs better than tree-based and DHT-based pub/sub. This difference is more in lower matching ratios where fewer brokers have matching subscriptions. This is because in tree-based approach in average an event is forwarded through higher number of brokers to reach subscribers. Similarly, in DHT-based approach multi-hop nature of routing results in higher dissemination traffic. The other fact that is shown in this figure is that by increasing the number of clusters, event dissemination traffic also increases. This is because of bigger ring size caused by larger number of clusters. Since the ring dissemination phase is broadcasting event to all ring member irrespective of their subscriptions, many brokers in the ring may receive the event while they have not subscribed to it. This increases the number of transmitted events. However, the difference in dissemination traffic reduces in higher matching ratios because there will be more brokers in the ring with matching subscriptions. This results in fewer brokers without matching subscriptions that receive an event. Also the graph shows that the generated traffic to mask failures in our approach still is less than the traffic in tree-based and DHT-based approaches. There is an exception case that with 20 clusters and 10% matching ratio, cluster-based pub/sub generates almost same amount of traffic as tree-based approach. This is because of higher traffic resulting from broadcasting content among 20 clusters.

Broker load. We assume the event dissemination load as the average number of messages that a broker processes during content dissemination. Figure 4.1 shows the average amount of load for different matching ratio and 500 publications. As it can be seen the average load increases by popularity of events and again our approach outperforms tree-based and DHT-based approaches. The difference is more considerable in lower matching ratios. The main reason for higher load on brokers in tree-based systems is that many brokers may participate in event dissemination by just forwarding events toward subscribers. This means despite these brokers are not interested in an event, they receive the event and process it and send it toward subscribers. In our method, on the other hand, the only place that a broker without having matching subscription may receive an event is in ring dissemination phase. This clearly justifies why broker load is lower for the setting with lower number of clusters which implies smaller ring size. Therefore, in cluster-based pub/sub, specially with fewer clusters, fewer number of brokers with no matching subscription participate in event dissemination which results in lower average load on brokers. The figure also shows that the failure masking algorithm does not significantly increase the load and still results in less load on brokers compared to tree-based and DHT-based pub/sub without considering any failures for them.

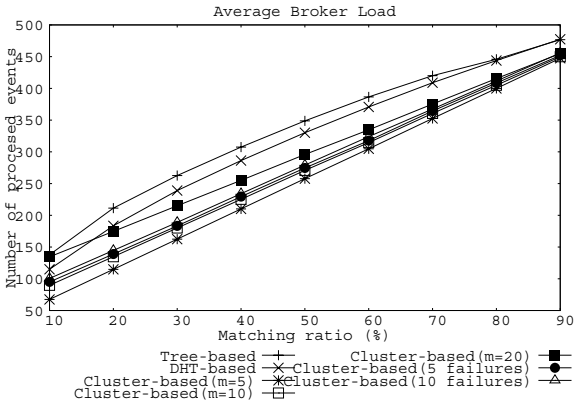


Figure 7. Average load on each broker.

5 Related Work

Most of content-based pub/sub systems that use tree-based content routing including [1, 2], assume a stable and

fault-free broker overlay network. However, some work tried to address the broker crash problem in tree-based pub/sub. Chand and Falber in [10] proposed a failure masking technique which is based on redundant routes where there is more than one route from publishers to subscribers. Since subscriptions and publications are forwarded through redundant paths, this approach provides higher availability to the cost of increasing already network traffic and broker load. [11] provides a broker failure masking mechanism using replicated brokers. In this mechanism, each node in the routing tree is a virtual broker containing a set of broker. Subscriptions are replicated in all brokers in a virtual broker while publication dissemination load is distributed among brokers in a virtual broker. When a broker crashes, the other brokers are used for forwarding event toward subscribers. However, using tree structure still degrades the content dissemination process and as shown in previous section results in higher network traffic and broker load. In [14], Baldoni et al. present a crash-resilient topology management technique which can recover from broker failure by requiring brokers to know subscriptions from their neighbor's neighbor. However, this method cannot address multiple broker crashes.

Most of pub/sub systems on structured overlay networks rely on the overlay network service for recovering from failures [5, 6, 12]. However, for preventing subscription lost these system use back up nodes for brokers to replicate subscriptions. In [6] each broker has a back up where is used when the primary broker crashes. However, if both a broker and its back up crash, subscriptions belonging to them will be lost.

In [13] a semi-probabilistic approach for event routing is introduced which is resilient to broker failures. Subscription propagation is limited to a few brokers. To forward events, each broker first uses the available information to find the route. If there is not useful information in subscription table, the event is randomly forwarded to subset of neighbors. However, the correct event routing is probabilistic and even in absence of broker failure, events may not be delivered to subscribers. Medym is another failure resilient approach introduced in [9]. In Medym subscriptions are broadcast to all brokers and when an event is published, its subscribers are determined in publisher broker and using a dynamic multicast algorithm event is propagated to them. However, broadcasting all subscriptions to all brokers results in high network traffic and subscription maintenance load on each broker.

6 Conclusions and Future Work

We presented a novel architecture for pub/sub based on clustering brokers and forming rings between these clusters. We showed how our proposed approach can mask broker failures and increase dissemination service availability. We also showed our approach outperforms tree-based and DHT-based pub/sub in dissemination time, traffic and load even when 10% of brokers crash in the system. The key insight enabling our approach to effectively mask failures is the replication of subscriptions in clusters and having multiple links between any pair of clusters.

We believe the proposed cluster-based architecture presents a novel pub/sub model that can be investigated regarding different factors. We are studying the effect of clustering based on different factors such as underlying network topology on the performance of the pub/sub system. We also aim to provide a hierarchical cluster-based architecture which can dynamically tune to different cluster settings based on subscription and event load.

References

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

[2] 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.

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

[4] 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.

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

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

[7] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, H. Balakrishnan, *Chord: a scalable peer-to-peer lookup protocol for internet applications.*, IEEE/ACM Trans. Netw. 11(1),2003.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, *A Scalable Content-Addressable Network.*, ACM SIGCOMM 2001.

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

[10] R. Chand and P. Felber, *XNet: A Relliable Content-based Publish/Subscribe System.*,IEEE SRDS, 2004.

[11] S. Bhola, R. Strom, S. Bagchi, Y. Zhao, J. Auerbach, *Exactly-once Delivery in a Content-based Publish-Subscribe System.*, DSN 2002.

[12] P. Pietzuch and J. Bacon, *Hermes: A Distributed Event-Based Middleware Architecture.*, DEBS02, 2002.

[13] P. Costa, G.P. Picco, *Semi-Probabilistic Content-Based Publish-Subscribe.*, IEEE ICDCS 2005.

[14] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito, *A Self-Organizing Crash-Resilient Topology Management System for Content-Based Publish/Subscribe.*, DEBS04, 2004.

[15] F.B. Schneider, *Byzantine generals in action: Implementing fail-stop processes*, ACM Trans. on Computer Systems, Vol. 2, 1984.

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

[17] J-Sim Simulator. http://www.j-sim.org/

[18] E. Zegura, K. Calvert and S. Bhattacharjee, *How to Model an Internetwork.*, IEEE Infocom 96.

[19] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz, *Tapestry: A Resilient Global-scale Overlay for Service Deployment*,IEEE JSAC, January 2004, Vol. 22, No. 1.