Efficient Content-Based Publish/Subscribe Systems over Peer-to-Peer Networks

Shou-Chih Lo

Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan
Email: sclo@mail.ndhu.edu.tw

ABSTRACT

Content-based publish/subscribe provides a convenient platform for users to specify their interested events as subscribers and ensure publications to be efficiently delivered to those users who are interested in the contents of them. Significantly, the current research trend is towards involving peer-to-peer network technology into this kind of system for scalability and fault tolerance. The dominant factor of system performance is to hash subscriptions and events into peer nodes for having time and space efficiency. In this paper, we propose two approaches with and without user clustering, respectively. Moreover, we introduce data replication and partition techniques to further distribute system workload. Our proposed system is more practical and flexible than existing ones. The performance is evaluated through a discrete event based simulator.

Keywords: Publish/Subscribe, Content-Based, Peer-to-Peer Network, User Clustering, Data Replication, Data Partition

1. INTRODUCTION

Publish/Subscribe (Pub/Sub) is an efficient communication paradigm for information dissemination in a large-scale distributed environment. A typical pub/sub system consists of *publishers*, *subscribers*, and *brokers*. Publishers, which act as information providers, publish events to brokers. Subscribers, which act as information consumers, express their interests on events by issuing subscriptions to brokers. As service nodes in the network, the functions of brokers are to store, deliver, and match of subscriptions and events.

Two types of pub/sub systems are given [12]: topic-based and content-based. In topic-based systems, publishers label each event with a topic (subject, group, or channel) and subscribers subscribe to all the events of certain topics. The major drawback of this type of system is that subscribers could not express their interests using fine-grained predicates. Content-based systems, on the other hand, allow of high expressive predicates over the contents of interested events.

Each content-based pub/sub system supports a predefined schema with a set of attributes over

which publishers set values to publish events and subscribers set predicates to filter events. For example, a subscriber can issue a subscription {Stock = IBM, Price < 95, Volume > 1000}, and a publisher can issue an event {Stock = IBM, Price = 90, Volume = 1200}. An event is said to match a subscription if all the attribute values set in the event satisfy the filter predicates specified in the subscription.

Generally, a subscription predicate is the association of a set of A_i θ V_i (where A_i represents an attribute name, V_i represents an attribute value, and θ is any relational operator) using conjunctive and/or disjunctive propositions. However, we can limit that a subscription predicate contains only conjunctive propositions. Any disjunctive proposition can be removed by dividing a subscription into several sub-subscriptions accordingly. An event predicate is the conjunctive association of a set of $A_i = V_i$.

In traditional pub/sub systems [1][5][6][7][8], the brokers are organized into a tree-shaped overlay network. These systems are simple but suffer from fault-tolerance problems. A new research trend is to introduce peer-to-peer (P2P) overlay network technology into the design of content-based pub/sub systems for having the benefits of scalability, load balancing, and fault tolerance [2][9][21][22][23]. Each subscriber host can act as a broker and all brokers are organized into a P2P overlay network. Subscription and event management is then viewed as a process of hashing a set of subscriptions and events into a set of brokers. Those brokers that are hashed to by at least one subscription or event are called rendezvous brokers (RBs). This hashing has to satisfy the so-called intersection rule [2]: For any matched subscription *s* to an event *e*, the set of RBs hashed by *e* intersects with the set of RBs hashed by *s*.

The motivations of our work are from two aspects. First, existing hash schemes can not achieve a low total cost on subscriptions/events delivery, match, and maintenance. Second, existing content-based pub/sub systems only consider a global system schema without the consideration to topic-based interests. Subscribers have to choose related attributes from the schema by themselves in specifying interests of certain topics. However, it would be convenient for subscribers on specifying their interests if there is a reference topic hierarchy. For example, some subscribers may have general interests in sports and related news. Though we can add a topic attribute in the system schema whose values can distinguish subscriptions of different topics, this topic attribute becomes a hot attribute which appears in all subscriptions. Hot attributes may lead to the load unbalancing problem on subscription and event distributions over an overlay network for most proposed hash schemes.

As a result, we need a new content-based system that also supports subscribers with topic-based interests and need an efficient hash scheme with low system cost as well. In this paper, two design approaches are discussed by considering a single or multiple clusters structure. To further improve performance, two advanced data placement strategies are introduced and compared: data replication and data partition. The remainder of this paper is organized as follows. Section 2 introduces background knowledge and related work. Section 3 illustrates our two proposed system approaches. Simulation study is conducted in Section 4. Finally, we draw a conclusion in Section 5.

2. PRELIMINARIES

P2P overlay networks have attracted much attention due to some excellent characteristics in a large-scale distributed environment. Some system protocols such as Chord [19], CAN [16], and Pastry [15] have been proposed to build large file sharing applications. These protocols are based on the DHT (distributed hash table) mechanism which allows shared files to be uniformly distributed into peers in a P2P network.

The Chord protocol is taken as our major reference in design. All peers (or nodes) in Chord are uniformly hashed into an identifier circular space (Chord space) by their IP addresses, and then they are connected into a basic ring structure (Chord ring). Each peer is associated with an identifier it is hashed into. Using the notation *peer*(k), we denote a peer with identifier k. A shared file is stored into one peer according to the hash value of a selected key (e.g., file name). A shared file with hash value k (denoted by *data*(k)) is stored into the successor peer of identifier k (denoted by *succ*(k)) which is the first peer encountered clockwise with its identifier larger than k. To speedup data search, each peer maintains a routing table (or finger table) to other neighboring peers. The three terms peer, node, and broker are used interchangeably in the rest of the paper.

The example operations of a pub/sub system over this Chord structure are shown in Fig. 1. Subscriptions s_1 and s_2 are delivered from their originated brokers to their successor brokers according to the hash values of attribute names “Price” and “Stock”, respectively. The delivery path is depicted with a solid line in the figure. Event e is delivered to three brokers according to the hash values of attribute names “Stock”, “Price”, and “Volume”. The delivery path is depicted with a dotted line in the figure. The delivery cost of a subscription or event is measured in terms of logical hops in the delivery path. The match cost of an event is measured in terms of the number of subscriptions that are examined with this event. The mean match latency of an event is the average time duration from the moment the event is published to the moment one subscription is found to

be matched with this event.

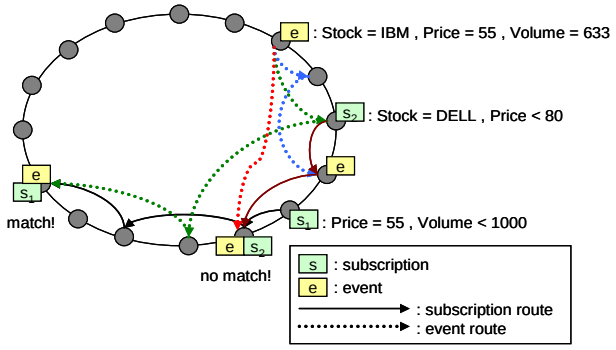


Fig. 1. Pub/Sub system operations over Chord.

The examples of topic-based pub/sub systems include Scribe [4], Corona [17], and FeedTree [20]. Scribe is built over the Pastry overlay network. The members (publishers and subscribers) of each topic group are connected into an individual multicast tree over the overlay network. The node that is hashed into by the topic name (or top identifier) acts as the root of the multicast tree. To reduce maintenance cost, a dynamic clustering technique is proposed in [12] to group topics with similar sets of subscribers into a virtual topic. Hermes [14] enables content-based filtering within a topic-based system by distributing filter expressions through the multicast tree.

The examples of content-based pub/sub systems include Ferry [23], eFerry [22], Meghdoot [9], and the proposed system by Baldoni et al. [2]. According to hash schemes employed, these content-based systems can be classified into two broad categories: attribute-name-based and attribute-value-based ones. An attribute-name-based scheme stores and delivers subscriptions and events upon the hash values of selected keys from attribute names (For example, we select Price as a key for s_1 in Fig. 1). While an attribute-value-based one performs these jobs upon the hash values of selected keys from attribute values (For example, we select 55 as a key for s_1 in Fig. 1). Ferry and eFerry belong to the attribute-name-based category. The work done by Baldoni et al. and Meghdoot belong to the attribute-value-based category. Except for Meghdoot which is built over CAN, the other systems are built over Chord.

Ferry stores a subscription into a broker by referring to the identifier number of the subscriber that issues the subscription. An event is sent to all RBs to guarantee the intersection rule. The RBs are those brokers that are hashed to by all attribute names in the schema. eFerry considers the combination of an arbitrary number of attribute names involved in the subscription and stores a subscription into a broker using the hash value of this combination. An event is then delivered to those brokers that are hashed to by each element in the power set of the set of attribute names in-

volved in the event.

Baldoni et al. propose a hash scheme that relies on the attribute values involved in a subscription or event. For a subscription, we randomly select one attribute and enumerate all attribute values in steps of precision units within the value range specified. Then this subscription is stored into those RBs that are hashed to by these attribute values. An event is hashed to those RBs by all attribute values specified in the event. Meghdoot stores a subscription to a single RB by translating range predicates into one point in a $2\times Y$ dimensional space (Y is the number of attributes in the schema). An event is delivered to a half number of RBs on average and only a half number of brokers function as RBs.

Assume that a pub/sub system has attribute schema X. The set of attribute names in X is denoted by X_{name}. Each attribute is associated with value domain D. The union of the value domains of all attributes in X is denoted by X_{value}. The cardinality and the power set of set X are denoted by $|X|$ and $power(X)$, respectively. The average attribute numbers specified in an event is n_e. The range predicate associated with an attribute in a subscription has an average range size of R. Assume that a pub/sub system has N brokers and the employed hash function is h. We can highlight a pub/sub system with descriptions: A subscription and an event are hashed to average m and n distinct RBs, respectively; there are totally k RBs in the system. The cost comparisons of existing content-based systems are listed in Table 1.

Table 1. Comparison of existing systems.

System	Average number of RBs that a subscription is hashed to (m)	Average number of RBs that an event is hashed to (n)	Total number of RBs (k)
Ferry	1	$\|h(X_{name})\|$	$\|h(X_{name})\|$
eFerry	1	2^{n_e}	$\|h(power(X_{name})\|$
Baldoni et al.	R	n_e	$\|h(X_{value})\|$
Meghdoot	1	$N/4$	$N/2$

The performance of these systems has been compared by us in another paper [11]. Basically, Ferry and eFerry suffer from high event delivery cost (proportional to value n), the proposed scheme by Baldoni et al. suffers from high subscription storing cost (proportional to value m), and Meghdoot has high maintenance cost due to using high dimensional space. In this paper, we will design a new hash scheme which has good performance on the sum of these cost metrics. The challenge is how to generate more RBs, distribute uniformly subscriptions and events into less

RBs, and reduce the number of match examinations.

3. PROPOSED SCHEMES

Two main approaches are proposed to manage subscriptions and events having topics specified. We first assume that all topics are organized into a hierarchy where only leaf topics are used. The removal of this restriction is discussed in Section 3.3. A subscriber belongs to topic group *T* if there is one subscription of topic *T* is issued by this subscriber. A subscriber can be a member of more than one topic group.

First is to consider system design with a single cluster. That is, all subscriber nodes are connected into a single Chord ring. Then, a hash scheme is proposed to balance subscription and event distributions. In this approach, all subscriber nodes share overall system workload. Another system design with multiple clusters is considered as well. Subscriber nodes belonging to the same topic group are connected into a virtual cluster structure. These virtual cluster structures are physically constructed over the same Chord ring. In this approach, a subscriber node would not store irrelevant data with high probability.

3.1. Single-Cluster Approach

In our previous work [11], a new hash scheme called Interval Index Mapping (IIM) is proposed. Here, we extend IIM by introducing topic groups and advanced data placement strategies.

IIM is an attribute-value-based scheme. Randomly, we select one attribute specified in a subscription (Section 4 examines different attribute selection methods). If the value domain of the selected attribute is [0, 500], for example, we divide this domain into 100 (a configurable number) equal-sized intervals like [0, 5), [5, 10), [10, 15), and so on. The starting values of these intervals are used as selected keys. If the value range specified by the selected attribute is [2, 8], for example, the subscription will be hashed to the RBs using keys 0 and 5 (since [2, 8] falls into intervals [0, 5) and [5, 10)). For an event, all attribute values specified in the event are considered to get their located intervals. Then, the event is hashed to the RBs using the starting values of these intervals.

To further achieve uniform distribution, we combine together on the hash the interval value, the selected attribute name, and the topic identifier a subscription/event belonging to. Let *Topic*(*T*) denote the identifier of topic *T*. All predicates specified in subscriptions or events are viewed as rang-type ones (i.e., using relation operations). Equal-type predicates have value ranges of size one.

For an attribute A_i, we have the following notations:

- $V_{\max}(A_i)$: maximal value in the value domain of A_i.
- $V_{\min}(A_i)$: minimal value in the value domain of A_i.
- *Interval*(A_i): size of an interval for the value domain of A_i.
- *Index*(v, A_i): interval number of value v in the value domain of A_i which is computed by $\left\lfloor \frac{v - V_{\min}(A_i)}{Interval(A_i)} \right\rfloor$.
- $V_{\text{high}}(A_i, s)$: highest value within the value range of A_i specified in subscription/event s.
- $V_{\text{low}}(A_i, s)$: lowest value within the value range of A_i specified in subscription/event s.
- *Name*(A_i): attribute name of A_i.

We perform subscription storing and event delivery by procedures SubscriptionStore and EventDelivery, respectively. As can be seen, a subscription is stored into more than one RB.

Procedure: *SubscriptionStore*(*s*, *T*)
//*CHORD_SIZE*: size of the Chord space
Begin
1. Randomly select an attribute A_i in subscription *s*.
2. $gap = CHORD\_SIZE / \left\lceil \frac{V_{\max}(A_i) - V_{\min}(A_i)}{Interval(A_i)} \right\rceil$.
3. $I = Index(V_{\text{low}}(A_i, s), A_i)$.
4. While ($I \le Index(V_{high}(A_i, s), A_i)$,
5. $S\_ID = (I \times gap + V_{\min}(A_i) + h(Name(A_i) + Topic(T))) \bmod CHORD\_SIZE$.
6. Store *s* into *succ*(*S_ID*).
7. $I = I + 1$.

End.

Procedure: *EventDelivery*(*e*, *T*)
Begin
1. For each attribute A_i in event *e*,
2. $gap = CHORD\_SIZE / \left\lceil \frac{V_{\max}(A_i) - V_{\min}(A_i)}{Interval(A_i)} \right\rceil$.
3. $S\_ID = (Index(V_{low}(A_i, e), A_i) \times gap + V_{\min}(A_i) + h(Name(A_i) + Topic(T))) \bmod CHORD\_SIZE$.
4. Deliver *e* to *succ*(*S_ID*).

End.

The basic IIM scheme could not guarantee load balancing among RBs, particularly when subscriber interests present skewness. The techniques of data replication and data partition are introduced and compared to solve this problem. The using of data replication is recommended in [18], where a subscription is replicated to other arbitrary $C-1$ brokers besides the originally hashed one. An event when hashed into one broker is examined locally or redirected to any one of these $C-1$ brokers. This significantly reduces the workload of event match on an RB but also incurs high subscription storing cost.

While using data partition, we divide the subscriptions stored in an RB into C segments. Only

one segment is remained and the other $C-1$ segments are distributed into other $C-1$ brokers with each broker storing one segment. In other words, a subscription is stored into one of these C brokers with equal probability. An event when hashed into one broker needs to be further forwarded to other $C-1$ brokers. This brings the advantage of parallel matching process among different brokers without incurring extra overhead on subscription storing. For using data replication, a network structure with high-connectivity requirement is adopted in [18]. However, we reuse the Chord structure with little overhead to involve these advanced data placement strategies. For each broker, we select other brokers of C-1 from connected neighbors by looking up the finger table. A broker which becomes overloaded if the number of stored subscriptions exceeds a threshold value will activate the data replication or partition process as shown below.

Procedure: Data Replication/Partition

Begin

1. Each broker keeps the states (alive, the number of stored subscriptions, etc.) of its neighboring brokers.
2. If a broker itself becomes overloaded, this broker distributes later coming subscriptions into at most $C-1$ neighboring brokers that are not overloaded according to either the replication or partition policy.

End.

3.2. Multi-Cluster Approach

In the multi-cluster approach, we group subscribers with similar interests into clusters. The members of each group manage their own data. The low cross correlation between clusters can prevent a subscriber node to store irrelevant subscriptions with high probability. Moreover, the cluster structure can facilitate the applications such as inner-group communications and recommendations. Diminished Chord [10] is one such structure that allows group members form their own Chord rings with least overhead. Conceptually, there is only one physical Chord ring which includes several virtual subrings and each of these sub-rings connects the members of each group.

A virtual subring is actually organized into a virtual binary tree. Consider the example of Fig. 2a where peers P_1, P_2, P_5, and P_6 join the same group. The binary tree is constructed by recursively dividing the Chord space into two equal-sized regions. In the first run, we have the whole space (denoted by region (0, 0] which covers *peer*(0)~*peer*(7)). In the second run, we get two equal-sized regions (0, 4] (which covers *peer*(1)~*peer*(4)) and (4, 0] (which covers *peer*(5)~*peer*(0)). Each region (x, y] has a representative region server which is *succ*(y). By keeping the parent-child relationship of these region servers, we have root node *succ*(0) which has two child nodes *succ*(4) and *succ*(0). This process continues until a tree with a depth of log$N$ (N is the

size of the Chord space) is constructed as shown in Fig. 2b.

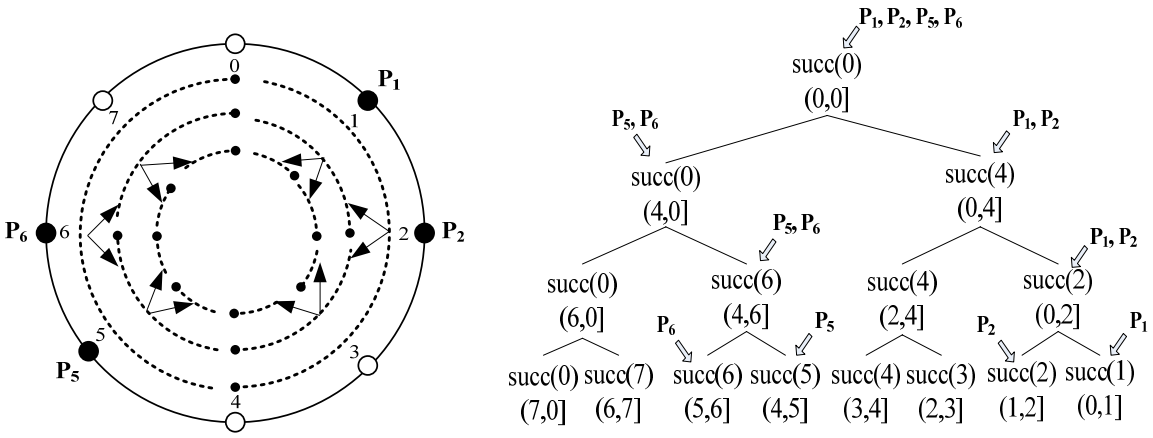


Fig. 2. Virtual binary tree.

Each *peer*(*k*) if having a physical mapping node registers its present to the leaf region server *succ*(*k*) and all its ancestor region servers in the tree. These registrations has annotated in Fig. 2b. For storing *data*(*x*), we find from the leaf region server *succ*(x) to the root region server until at least one registered peer is found. We denote the set of registered peers in the final located region server as *RPeers*. Then, *data*(*x*) is stored into *succ*(*x*, *RPeers*) which returns the successor peer of *x* by assuming that only peers in *RPeers* remain within the Chord ring. For example, the final located region server is *succ*(4) at level 1(the root is at level 0) for storing *data*(4). Among the two registered peers (P_1 and P_2), we store *data*(4) to *succ*(4, $\{P_1, P_2\}$) which returns P_1. The finding of *data*(*x*) follows the same path of storing *data*(*x*).

Diminished Chord presents three phenomena:

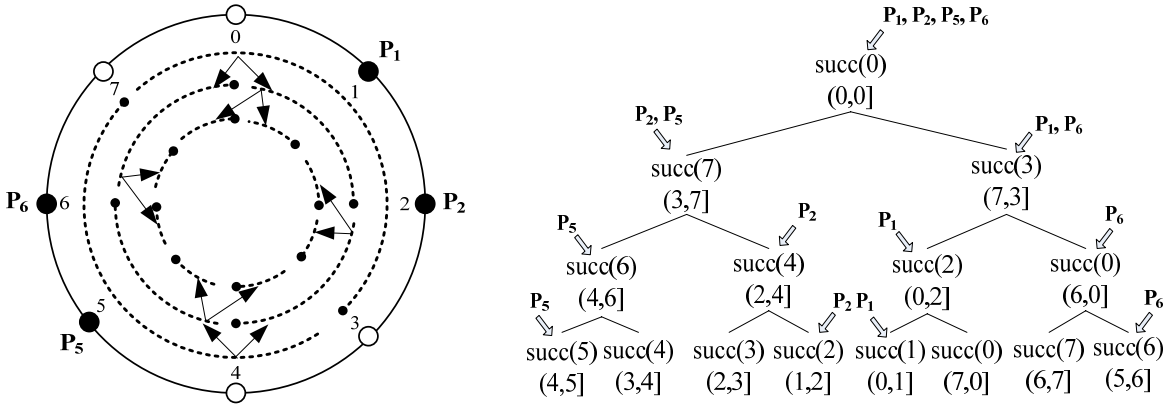
1. The tree depth is fixed to the maximal length of log*N*.
2. A peer may act as a server for different regions simultaneously in the same tree.
3. Shared files may not be evenly distributed into registered peers.

Phenomenon 1 becomes inflexible to dynamic changing environments. Phenomenon 2 is because that each left child node is the node itself as shown in Fig. 2b. This property has a tradeoff that the physical path length from a leaf node to the root node might be shorter than the tree depth but some region servers might become overloaded. Phenomenon 3 is because that shared files hashed into a region server where there is no any registered peer are stored into the same peer. For example, all shared files hashed into *peer*(3) and *peer*(4) are stored into P_1.

Here we provide improvements on these phenomena. Phenomenon 1 can be easily relaxed by

stopping tree generation to a given depth. To moderate the influences of phenomena 2 and 3, we introduce rotation and remap operations, respectively. In tree construction from top to down and level by level, we continue rotating each divided region counterclockwise with distance of r ($r \leq N/\log N$) in the Chord space. A region server of $succ(x)$ at level k is rotated as one of $succ((x-kr+N) \bmod N)$. Fig. 3a shows the same example of Fig. 2a but with rotation distance of 1. Fig. 3b shows the constructed tree. In the comparison, the tree of Fig. 3b is more balanced than that of Fig. 2b, since peers are more uniformly registered to region servers.

In the previous example of Fig. 2b, we locate $succ(4)$ at level 1 in the tree for storing data(4), and finally do the storing to $succ(4, \{P_1, P_2\})$. We can find that the registered peers P_1 and P_2 to $succ(4)$ are all from the right child node (i.e., $succ(2)$) and this child node has the responsible region (0,2]. To uniform distribute data storing to these registered peers, we perform the remap operation that re-hashes the key value of the stored data into region (0,2]. Consequently, data(4) is remapped to data(2) and is stored to $succ(2, \{P_1, P_2\})$ which returns P_2. Compared with the case in Fig. 2b, all shared files hashed into $peer(3)$ and $peer(4)$ are stored into P_1 and P_2, respectively.



(a) (b)

Fig. 3. Virtual binary tree with rotation.

In our multi-cluster system, each topic T in topic hierarchy has its own virtual binary tree (called group T tree). For any $data(x)$ or $peer(x)$ of topic T, we can compute the identifier of the region server at level k in the group T tree whose responsible region covers value x by function *get_RegionServer*. As can be seen, the computation does not traverse physical tree links, and hence we need not maintain the tree structure.

get_RegionServer(Topic T, Level k, Value x)
//*RORATE_DIST*: rotation distance
//*CHORD_SIZE*: the size of the Chord space
{ $topic\_hash = (h(Topic(T)) - k \times RORATE\_DIST + CHORD\_SIZE) \bmod CHORD\_SIZE$;
$region\_size = CHORD\_SIZE/2^k$;

relative_hash = *x*−*topic_hash*;
if (*relative_hash* < 0) *relative_hash* = *relative_hash* + *CHORD_SIZE*;
server_hash = ((*relative_hash*/*region_size* + 1)×*region_size* + *topic_hash*) mod *CHORD_SIZE*;
return *server_hash*;
}

The following describes the operations of node join and leave, subscription storing, and event delivery.

Procedure: *Join*(*x*, T) //The join procedure of node *x* into group *T* tree.
Begin
1. Perform the basic join operation into Chord if node *x* is not in the Chord ring.
2. Locate the root node of group *T* tree that is *succ*(*get_RegionServer*(*T*, 0, *x*)) and retrieve the basic settings about the group tree such as the rotation distance and tree depth.
3. For level *k* = 0 to *depth*(*T*)
4. Register node *x* to the region server *succ*(*get_RegionServer*(*T*, *k*, *x*)) if node *x* is not yet registered.
5. Transfer the relevant data of group *T* that are now under the charge of node *x* from the immediate successor node.

End.

Procedure: *Leave*(*x*, *T*) //The leave procedure of node *x* from group *T* tree.
Begin
1. For level *k* = 0 to *depth*(*T*)
2. De-register node *x* from the region server *succ*(*get_RegionServer*(*T*, *k*, *x*)).
3. Transfer all relevant data of group *T* to the immediate successor node.
4. Perform the basic leave operation if node *x* wants to leave the Chord ring as well.

End.

Both join and leave procedures involve the registration or de-registration process to the region server of each level of a group tree. The cost to locate a region server is O(log*N*) and the depth of a group tree is O(log*N*), so the additional cost of node join and leave is O($\log^2 N$) for normal operations in Chord. Each node, in addition to periodically refresh the finger table, should refresh a registration table with at most O(log*N*) entries which record registered region servers.

The similar IIM scheme is deployed in the multi-cluster approach as follows:

Procedure: *SubscriptionStore*(*s*, *T*)
Begin
1. Randomly select an attribute A_i in subscription *s*.
2. *gap* = $CHORD\_SIZE / \left\lceil \frac{V_{\max}(A_i) - V_{\min}(A_i)}{Interval(A_i)} \right\rceil$.
3. *I* = *Index*($V_{\text{low}}(A_i, s)$, A_i).
4. While (*I* ≤ *Index*($V_{high}(A_i, s)$, A_i),
5. *S_ID* = (*I*×*gap*+$V_{\min}(A_i)$+*h*(*Name*(A_i))) mod *CHORD_SIZE*.
6. For level *k* = *depth*(*T*) to 0,
7. *Region_Server* = *succ*(*get_RegionServer*(*T*, *k*, *S_ID*)).
8. *RPeers* = the set of registered nodes in *Region_Server*.
9. If *RPeers* is not empty,
10. If *k* ≠ *depth*(*T*), *S_ID* = *ReMap*(*T*, *k*, *S_ID*).
11. Store *s* into *succ*(*S_ID*, *RPeers*).
12. Exit the for loop.
13. *I* = *I*+1.

End.

Procedure: *EventDelivery*(e, *T*)
Begin
1. For each attribute A_i in event *e*,
2. $gap = CHORD\_SIZE / \left\lceil \frac{V_{\max}(A_i) - V_{\min}(A_i)}{Interval(A_i)} \right\rceil$.
3. $S\_ID = (Index(V_{low}(A_i, e), A_i) \times gap + V_{\min}(A_i) + h(Name(A_i))) \bmod CHORD\_SIZE$.
4. For level $k = depth(T)$ to 0,
5. $Region\_Server = succ(get\_RegionServer(T, k, S\_ID))$.
6. *RPeers* = the set of registered nodes in *Region_Server*.
7. If *RPeers* is not empty,
8. If $k \neq depth(T)$, $S\_ID = ReMap(T, k, S\_ID)$.
9. Deliver *e* into $succ(S\_ID, RPeers)$.
10. Exit the inner for loop.

End.

The delivery of a subscription or event follows the same process flow[1] along the path from a leaf region server to the root node of a group tree until there is any registered node found. This takes at most O(log*N*×log*N*) delivery cost in logical hops. Note that the traversal of a sequence of region servers during this process always follows the clockwise direction in the Chord space, which facilitates a series of successor operations. This is because we perform rotation operations along the opposite (i.e., counterclockwise) direction. If we finally locate at a non-leaf region server, the *ReMap* procedure is performed, and then we select one registered node to do subscription storing or event matching. A subscriber should periodically reissue its subscriptions into the system just after the rejoin procedure to handle any topology change.

```
ReMap(Topic T, Level k, Value x)
{ topic_hash = (h(Topic(T))+(k+1)×RORATE_DIST) mod CHORD_SIZE;
  region_size = CHORD_SIZE/2^(k+1);
  relative_hash = x–topic_hash;
  if (relative_hash < 0) relative_hash = relative_hash+CHORD_SIZE;
  from_region = relative_hash/region_size;
  if (from_region is an even number) re_hash = x+region_size;
  else re_hash = x–region_size;
  if (re_hash < 0) re_hash = re_hash+CHORD_SIZE;
      return re_hash;
}
```

The same data replication and partition techniques in the single-cluster approach can be applied here. However, we distribute subscriptions among the registered nodes to the same region server for keeping the property of low cross correlation.

3.3. General Considerations

Our system design can be further discussed from three aspects: match semantics, topic hierar-

[1] A publisher when issuing an event may first contact with the root of a group tree to retrieve information about basic tree settings.

chy, and event description.

Match semantics

Two match semantics are defined as exact match and partial match, and we assume that all predicates specified in a subscription or event are conjunctively associated. Let $P_s(A_i)$ and $P_e(A_i)$ denote the predicate specified on attribute A_i in a subscription or event, respectively.

Definition: An event *e* is said to exactly match subscription *s*, if for all attributes A_i specified in *s*, $P_e(A_i)$ satisfies with $P_s(A_i)$.

Definition: An event *e* is said to partially match subscription *s*, if for all attributes A_i specified in *s*, either $P_e(A_i)$ satisfies with $P_s(A_i)$ or $P_e(A_i)$ is not specified.

For example, if we have *e*: $\{A_1=10\}$ and *s*: $\{ A_1<20, A_2>5\}$, *e* partially matches with *s* but does not exactly match with *s*. Only Meghdoot applies partial match as we know and returns more results to subscribers than the other systems. Our system design follows the semantics of exact match. However, we can easily extend our system to allow partial match by handling all attributes specified in a subscription instead of only one selected attribute.

Topic hierarchy

At this aspect, we allow subscribers or publishers to annotate their data with non-leaf (or generalized) topics. A generalized subscription or event is substituted by underlying specialized ones in topic hierarchy. For example, a generalized subscription of topic *T* is substituted by subscriptions of those leaf topics in the subgraph rooted at topic *T* in the hierarchy. The side effect of this method is that a subscriber with generalized subscriptions would register itself to several group trees. To reduce this effect, a distributed merging process is provided.

Notice that the root of each group tree records all its member nodes. We let the root node periodically probe some other randomly selected root nodes and exchange the list of member nodes with each other. Then, we merge those topics T_i'*s* with similar node lists into a super topic T_s under the criterion that the number of equal members is greater than a threshold and the number of unequal members is less than a threshold. This super topic may either already exist in the hierarchy or does not exist. In the latter case, we generate a virtual topic whose identifier is combined by the identifiers of merged topics. The root of each group T_i tree puts a reference to the root of group T_s tree.

In join procedure, we redirect the registration to the group tree that is referred to. A subscriber or publisher remembers what group tree it is referred to and delivers subscriptions or events of topic T_i to this group tree. In an extreme case, all leaf topics merge into one super topic and the

system becomes a single-cluster one. When the merge criterion is violated, the root of a super topic will inform all roots of merged topics to remove their references.

Event description

Most existing content-based pub/sub systems allow only equal operations to be used in event description. Some location-based pub/sub systems [3] address the issue that an event can be specified with an area constraint. Any subscriber when moving into this area is notified with this event. For flexibility, any rational operation should be used in event description as in subscription description. Certainly, our proposed IIM scheme can handle this situation by treating event delivery in the same way as subscription delivery.

4. SIMULATION STUDY

We evaluate system performance through simulations written in CSIM [24] and measure two cost metrics: mean match latency and management cost in terms of time consumed. The management cost includes subscription storing, event delivery, match cost, and subscriber registration. Since the basic scheme has been compared with others in our previous work [11], we continue studying here the enhanced schemes themselves.

4.1. Environmental Settings

We consider a system with 512 subscriber nodes by default in a Chord space of size 4096. We totally generate 3000 subscriptions and then 9000 events into the system. Ten topic groups are used with each group associating with six attributes which are randomly selected from a system schema with 20 attributes. Each attribute in the schema has the same integer-type value domain from 0 to 99 which is further divided into six equal-sized intervals. The mean time to transfer a subscription or event over a logical hop is 2.0 sec.

A subscription is generated as follows. The belonging topic is selected according to the Zipfian distribution with parameter θ ($0 \le \theta \le 1$, 0.6 by default). The number of attributes involved is set according to a uniform distribution (The mean is three and the variance is one). Each attribute has a value range with the size controlled by a uniform distribution (The mean is 20 and the variance is five). The attribute for subscription storing is either randomly selected (called *random attribute selection*) or is selected with the probability inversely proportional to the size of its associated value range (called *proportional attribute selection*). The latter selection scheme is a default ones in our experiments.

An event is generated as follows. We randomly select one subscription generated before to be the matched subscription of this event. The event then has attribute values that are randomly given but are satisfied with the constraint of the matched subscription. The event inter-arrival time follows an exponential distribution.

Two workload environments are distinguished including busy and free ones. Te corresponding parameter settings are listed in Table 2.

Table 2. Two workload environments.

	Busy environment	Free environment
Mean event inter-arrival time	1.0 sec	2.0 sec
Time to store/examine a subscription in a broker	0.01 sec	0.001 sec
Time to look up routing information in a broker	0.01 sec	0.001 sec

4.2. Experimental Results

4.2.1 Performance of the single-cluster approach

To begin with the single-cluster approach, we compare two attribute selection schemes: random and proportional ones under a free environment. Fig. 4a shows that the management cost of using the proportional scheme is slightly less than that of using the random one. Fig. 4b shows the small difference on match latency between these two schemes. For the further study, the experiment is performed again by changing the variance of the range size distribution. As shown in Figs. 4c and 4d, the proportional scheme has a lower cost when the variance is getting large. This reveals that the random scheme might select an attribute with a large value range so as to produce many replicated subscriptions. No doubt, we get the same observation in a busy environment. The performance value with match latency may be vibrated particularly in small view scope as the one shown in Fig. 4d. The reason is that the hashing behavior in our experiments is not perfect.

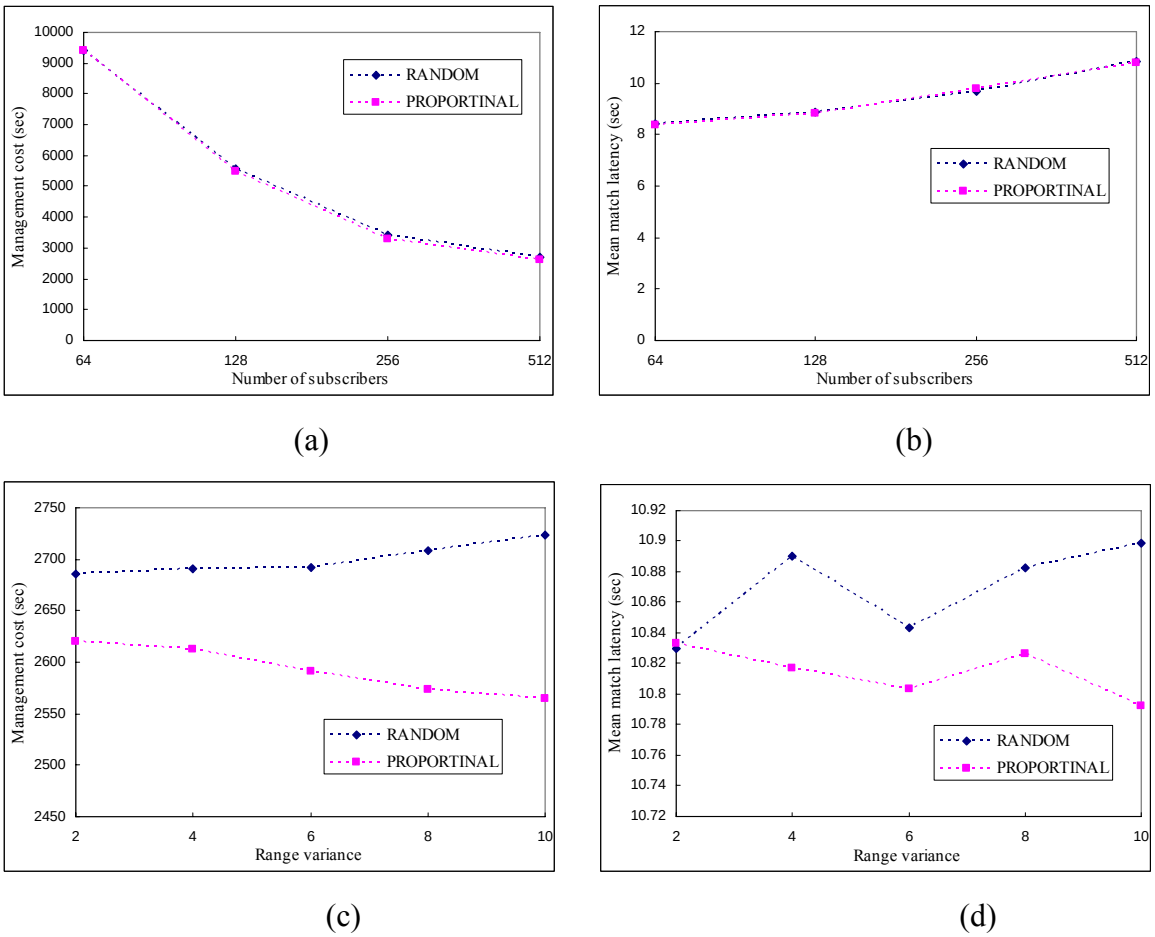


Fig. 4. Comparisons of attribute selection schemes.

What is more, the influence of different data placement strategies is observed on the performance. The finding demonstrates that using these strategies can improve performance only under busy environments. In the experiment, we further reduce the number of subscriber nodes to 128 to burden the workload on a broker. As aforementioned, a broker with the number of stored subscriptions exceeding a threshold value will activate one data placement strategy. The threshold is optimal when the total management cost is minimal. Fig. 5a shows the optimal threshold under different values of *C*. The threshold value becomes large as *C* increases, since too many replicated brokers bring negative effect. Using data partition has the lowest management cost when $C = 5$ as shown in Fig. 5b and slightly performs better than using data replication on match latency as shown in Fig. 5c. We found that using these data placement strategies can largely reduce match latency but may increase management cost with an arbitrary *C* value. Figs. 5c and 5d show the small difference (less than 2.0 sec) on match latency for a certain data placement strategy if using different *C* and threshold values. In other words, the optimal settings become less important if the design goal is for minimizing match latency.

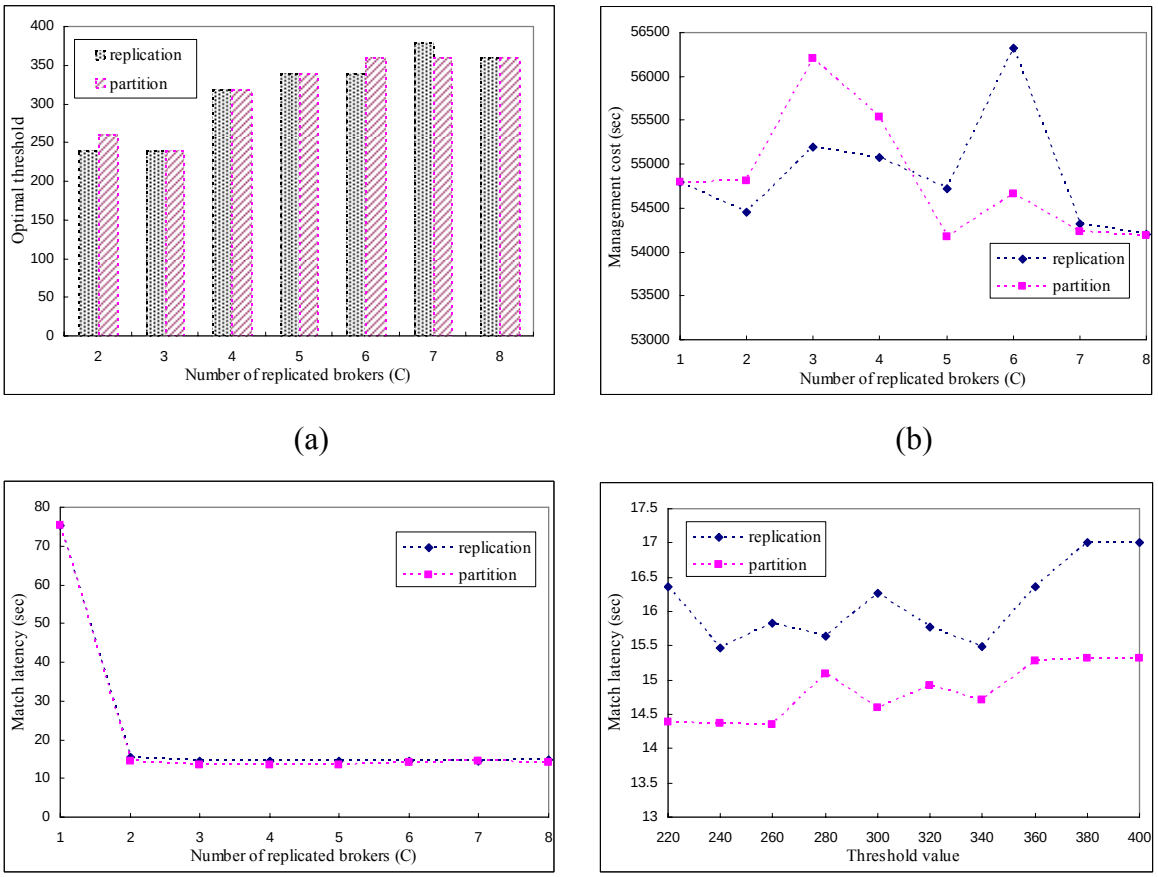


Fig. 5. Comparisons of data placement strategies.

4.2.2 Performance of the multi-cluster approach

First, we observe the influence of tree depth on performance. While the rotation distance is fixed to 250, as shown in Figs. 6a and 6b, the performance gets improvement with a deep tree particularly in the busy environment. Fig. 6c shows the optimal tree depth with the minimal management cost under different data skewness degrees. The average tree depth for 512 subscriber nodes is larger than that for 128 subscriber nodes. Actually, there is a tradeoff in having a deep tree. The negative effect is that more registration cost incurs and subscriptions or events take more hops to final destinations. The positive effect is that subscriptions or events get more chance to be rotated into a more uniform distribution state such that the number of RBs is increased and event match cost is decreased.

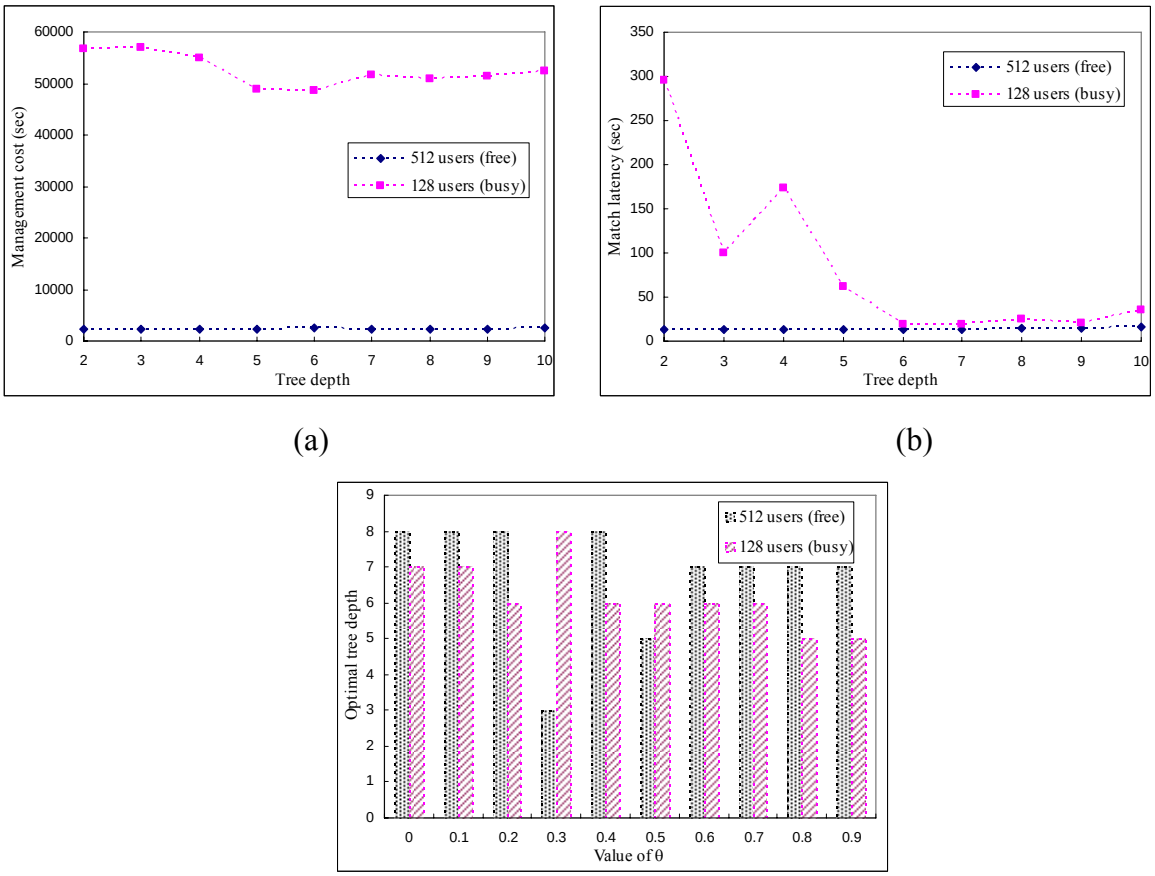
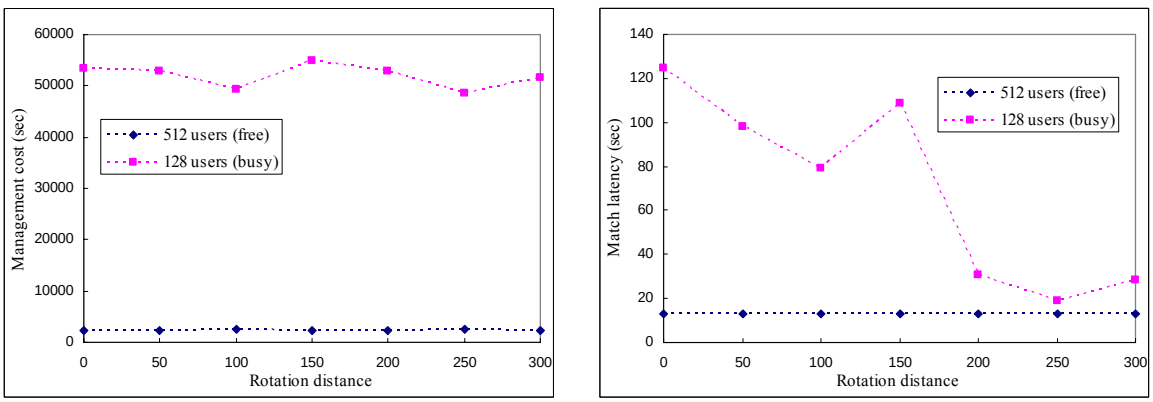


Fig. 6. The influence of tree depth.

Next, the influence of rotation distance is observed under two workload environments. The tree depth is fixed to six. The main goal of using rotation is for evenly distributing registered peers to region servers. Figs. 7a and 7b show that performance improvement becomes more significant in the busy environment. However, without a proper rotation distance, the distribution of registered peers might become skewed which degrades the performance. We propose an approach of periodically changing the rotation distance which is performed at the root node of each group tree.

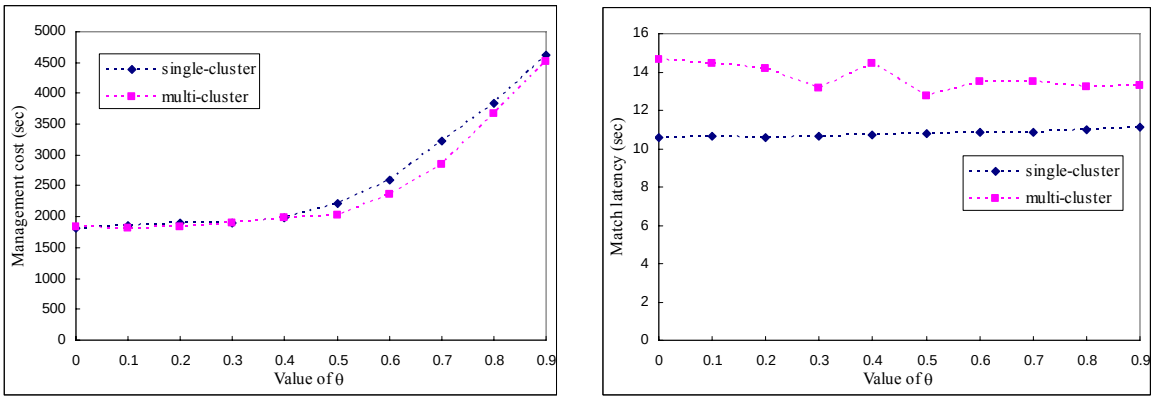


(a) (b)

Fig. 7. The influence of rotation distance.

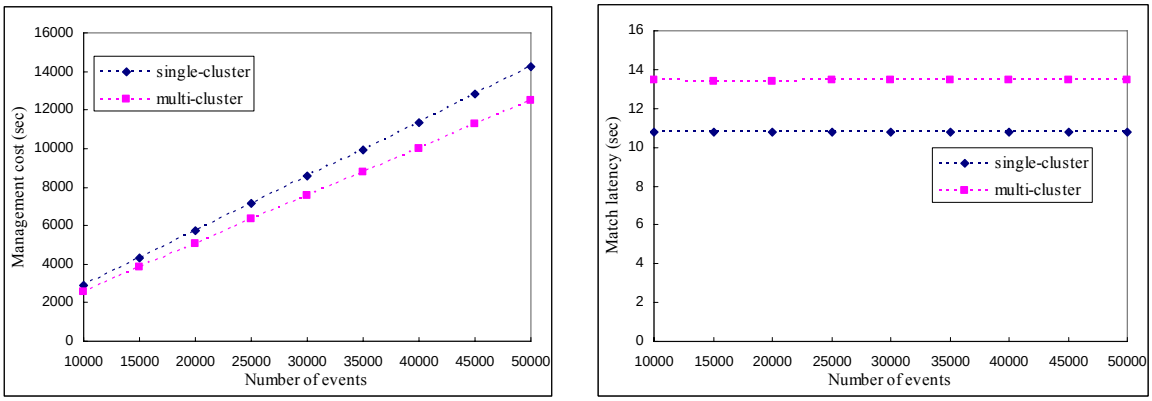
4.2.3 Comparison between single-cluster and multi-cluster approaches

The performance between single-cluster and multi-cluster are compared under the free environment. As shown in Figs. 8a and 8b, the multi-cluster approach achieves space efficiency due to having low management cost particularly under skewed data distribution ($\theta \geq 0.4$), and the single-cluster approach achieves time efficiency due to having low match latency. The skewed data distribution means that most subscribers are interested in events of hot topics and most publishers generate events of hot topics as well. Basically, the subscription and event delivery cost in the multi-cluster approach is higher than that in the single-cluster approach due to having longer route paths. This explains why the multi-cluster approach has long match latency. However, we found that more RBs are generated in the multi-cluster approach under the skewed data distribution due to rotation and remap operations. This makes the multi-cluster approach to have low match cost and reduces the total management cost. Figs. 9a and 9b show that the multi-cluster approach is excellent in space efficiency under a large volume of generated events and keeps the same gap with the single-cluster one on match latency.



(a) (b)

Fig. 8. Comparisons under different topic skewness degrees.



(a) (b)

Fig. 9. Comparisons under different event numbers.

5. CONCLUSIONS

The pub/sub system is a very appealing interaction model, and its capability can be fully enhanced over a P2P overlay network. The performance of this kind of system is dominated by the developed hash scheme. In the paper, we use the IIM hash scheme over two proposed architectures to provide both topic-based and content-based services. Also, we evaluate the possible improvement of using data replication or data partition on performance. These advanced data placement strategies bring great improvement on match latency under busy environments where there are many pending jobs for a subscriber node and these jobs take long time to process.

To sum up, the single-cluster architecture can achieve time efficiency in match latency, which is more suitable for time-critical environments. Additionally, the multi-cluster architecture can achieve space efficiency in data processing, which is more suitable for resource-constraint environments. In the future, we will extend this work to location-dependent service and mobile ad hoc network environments. Both subscribers and publishers can roam through the whole network. The new challenges include the location tracking, filtering of location-dependent criteria, and handling of node mobility.

REFERENCES

[1] G. Banavar, *T*. Chandra, B. Mukherjee, and J. Nagarajarao, "An Efficient Multicast Protocol for Content-Based Publish-Scribe Systems", *Proc. 19th IEEE Intl Conf. on Distributed Computing Systems*, pp. 262–272, 1991.

[2] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg, "Content-Based Publish-Subscribe

over Structured Overlay Networks," *Proc. 25th IEEE ICSCS*, 2005.

[3] G. Cugola and J. E. M. D. Cote, "On Introducing Location Awareness in Publish-Subscribe Middleware," *Proc. IEEE Intl. Conf. on Distributed Computing Systems Workshops*, 2005.

[4] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," *IEEE JSAC*, vol. 20, no. 8, pp. 1489-1499, 2002.

[5] G. Cugalo, E. D. Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS", *IEEE Trans. on Software Engineering*, vol. 27, no. 9, pp. 827-850, Sept. 2001.

[6] A. Carzagina, M. J. Rutherford, and A. L. Wolf, "A Routing Scheme for Content-Based Networking", *Proc. IEEE INFOCOM*'04, vol. 2, pp. 918-928, Mar. 2004.

[7] F. Y. Cao and J. P. Singh, "MEDYN: Match-Early and Dynamic Multicast for Content-based Publish/Subscribe Service Networks", *Proc. Fourth International Workshop on Distributed Event-Based Systems*, pp. 370–376, Jun. 2005.

[8] A. Carzagina and A. L. Wolf, "Forwarding in a Content-Based Network", *Proc. ACM SIGCOMM*'03, pp. 163-174, Aug. 2003.

[9] A. Gupta, O.D. Sahin, D. Agrawal, and A. E. Abbadi, "Meghdoot: Content-Based Publish/Subscribe over P2P Networks," *Proc. 5th ACM/IFIP/USENIX Intl. Conf. on Middleware*, pp. 254-273, Oct. 2004.

[10] D. R. Karger and M. Ruhl, "Diminished Chord: A Protocol for Heterogeneous Subgroup Formation in Peer-to-Peer Networks," *Lecture Notes in Computer Science*, vol. 3279, pp. 288-297, 2005.

[11] S.C. Lo and Y. T. Chiu, "Design of Content-Based Publish/Subscribe Systems over Structured Overlay Networks," *IEICE Trans. on Information and Systems*, vol. E91-D, no.5, May 2008.

[12] Y. Liu and B. Plale, "Survey of Publish Subscribe Event Systems," *Technical Report* TR-574, Indiana University, Computer Science Dept., May 2003.

[13] T. Milo, T. Zur, E. Verbin, "Boosting Topic-Based Publish-Subscribe Systems with Dynamic Clustering," *ACM SIGMOD*, pp. 749-760, 2007.

[14] P. R. Pietzuch and J. M. Bacon, "Hermes: A Distributed Event-Based Middleware Archi-

tecture," *Proc. Intl. Conf. on Distributed Computing Systems Workshops*, 2002.

[15] A. Rowstron and P. Druschel, "Pastry: Scalable Distributed Object Location and Routing for Large-Scale peer-to-Peer Systems," *Proc. IFIP/ACM Intl. Conf. on Distributed Systems Platforms* (Middleware), Nov. 2001.

[16] S. Ratnasaour, P. Francis, M. Handley, and R. Karp, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM*'01, vol. 31, no. 4, pp.161-172, Aug. 2001.

[17] V. Ramasubramanian, R. Peterson, and E. G. Sirer, "Corona: A High Performance Publish-Subscribe System for the World Wide Web," *Proc. of Networked System Design and Implementation*, 2006.

[18] C. Raiciu, D. S. Rosenblum, and M. Handley, "Revisiting Content-Based Publish/Subscribe," *IEEE Intl. Conf. on Distributed Computing Systems Workshops* (ICDCSW), pp. 19-24, 2006.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," *Proc. SIGCOMM*'01, 2001.

[20] D. Sandler, A. Mislove, A. Post, and P. Druschel, "FeedTree: Sharing Web Micronews with Peer-to-Peer Event Notification," *Proc. Intl. Workshop on Peer-to-Peer Systems*, 2005.

[21] P. Triantafillou and I. Aekaterinidis, "Content-based Publish-Subscribe over Structured P2P Networks," *Intl. Conf. on Distributed Event-Based Systems*, 2004.

[22] X. Yang, Y. Zhu, and Y. Hu, "Scalable Content-Based Publish/Subscribe Services over Structured Peer-to-Peer Networks," 15th Euromicro Intl Conf. on Parallel, Distributed and Network-Based Processing, pp. 171-178, 2007.

[23] Y. Zhu and Y. Hu, "Ferry: An Architecture for Content-Based Publish/Subscribe Services on P2P Networks," *Proc. of the ICPP*'05, 2005.

[24] The Mesquite Software Inc., "The User's Guide of CSIM Simulation Engine," http://www.mesquite.com/.