Small-Scale Peer-to-Peer Publish/Subscribe

Vinod Muthusamy[†] and Hans-Arno Jacobsen[†‡]
Middleware Systems Research Group
[†]Department of Electrical and Computer Engineering
[‡]Department of Computer Science
University of Toronto
{vinod,jacobsen}@eecg.toronto.edu

Abstract

The scalability of publish/subscribe (pub/sub) systems and distributed hash tables (DHTs) have been extensively studied in the literature. However, less well-known are properties of the pub/sub model and DHTs that make them suitable for small networks. This paper articulates these benefits, and evaluates the performance of a DHT-based pub/sub implementation in small-scale networks. We find that a fundamental assumption of DHT-based data management applications is violated in small networks, and this makes the pub/sub implementation exhibit poor load balance under certain workloads. This work illustrates that data management applications that scale in large networks may not scale in small networks.

1 Introduction

The publish/subscribe (pub/sub) [7, 6] model, with proven scalability and decoupling properties, is well suited to large-scale Internet applications that require selective data dissemination. There has been much research into both centralized pub/sub matching algorithms [7, 23] and distributed pub/sub routing protocols [2, 1]. Distributed hash tables (DHTs) [21, 19] have emerged as a routing substrate for large-scale dynamic networks. The large-scale benefits of pub/sub systems and DHTs have led to several implementations of DHT-based pub/sub systems [22, 3, 18, 8]. These implementations address a challenging problem since the queries, or subscriptions, that need to be managed by a pub/sub system do not fit naturally over the DHT interface. The difficulty lies in the fact that DHTs provide exact match lookup semantics, while content-based subscriptions may specify a *range* of interest. The first contribution of this paper is the design of a content-based pub/sub algorithm over a DHT interface.

The large-scale benefits of pub/sub and DHTs are well understood. However, less well-known are the properties of these systems that make them suitable for small networks. For example, the loose coupling of entities in the pub/sub model encourages a reusable software architecture, and DHTs are ideal to operate with error-prone, cheap, commodity hardware affordable in small networks.

It can be argued that in small systems, centralized architectures are sufficient, and may perform better than distributed protocols with their associated overhead costs. However, deploying a small-scale network based on protocols that scale to large networks allows the system to grow over time (simply by adding more machines to the network) without requiring changes to the architecture or protocols.

The small-scale benefits of pub/sub are somewhat articulated with Service Oriented Architectures [13, 4], but the small-scale advantages of DHTs are rarely considered. As another contribution of this paper, we identify the small-scale benefits of pub/sub and DHTs, and we evaluate the performance of our DHT-based pub/sub system in small networks, to determine if the proven large-scale performance benefits of DHTs and pub/sub systems apply in small networks.

After a quick background on the pub/sub model and DHT networks in Section 2, Section 3 articulates the properties of these systems that are advantageous in small-scale networks. In Section 4, the distributed pub/sub matching algorithm is developed, and Section 5 evaluates the algorithm. Section 6 describes some related work, and Section 7 completes the paper with some concluding remarks and discussion of future work.

2 Background

In order to keep the paper self-contained, this section provides a brief background on the pub/sub model and P2P networks.

2.1 Publish/Subscribe

Pub/Sub is a data dissemination model with three entities: the data producer, or *publisher*, sends data using *publication* messages, the data consumer, or *subscriber* expresses interest in the publications using *subscription* messages, and one or more *brokers* mediate between the two.

In content-based pub/sub, publications are a set of $\{$attribute, value$\}$ pairs, and subscriptions are a conjunction of $\{$attribute, operator, value$\}$ tuples, where the operator can be one of $\{=, <, >, \leq, \geq\}$. This allows subscriptions to discriminate based on the *content* of the publications.

2.2 Peer-to-Peer

P2P networks are characterized by the direct sharing of resources among the peers in the network, and can be loosely classified as unstructured and structured protocols. Unstructured networks such as Gnutella and Kazaa provide no performance guarantees. Structured networks [21, 19], often based on a distributed hash table (DHT) interface, are load balanced, fault-tolerant, and provide statistical guarantees on routing lengths and storage load.

A DHT stores (key,value) pairs in a network of nodes. The core operation of a DHT protocol is to map a key to a node and efficiently route messages to this node. In the Pastry [21] DHT, keys and nodeids are a sequence of 2^b digits and belong in a circular 128-bit identifier space, and are generated by the SHA-1 cryptographic hash. Pastry maps keys to the node with the numerically closest nodeid in the identifier circle, with the hash function ensuring good load balance: in a network with N nodes and K keys, each node stores K/N keys with high probability. Pastry uses a prefix routing algorithm that requires that each hop of a message is sent to a node that matches the destination nodeid by at least one more digit. This routing algorithm is able to route a message to the destination in $\lceil \log_{2^b} N \rceil$ overlay hops, and requires $O(\log N)$ node state.

3 Small-Scale Benefits

Existing research on pub/sub networks and DHTs has almost exclusively focused on how these systems scale to large networks. However, there are rarely considered and seldom exploited properties of both pub/sub and DHTs that make them suitable for the development of small-scale distributed applications. The sections below articulate some of these untapped small-scale benefits.

3.1 Publish/Subscribe Benefits

Some of the purported benefits of the pub/sub model are *multicast event dissemination* and *distributed matching*.

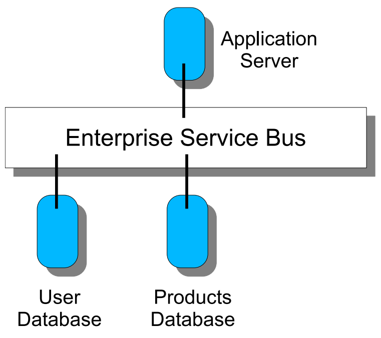


Figure 1. SOA architecture of a web server

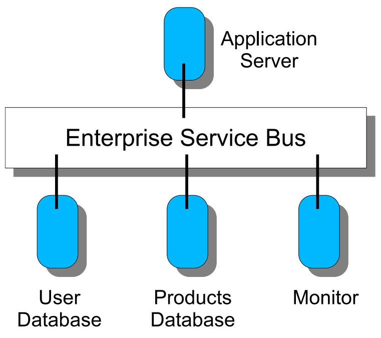


Figure 2. Pluggable monitoring service

The former enables one to many communication with sublinear message scalability, and the latter allows each broker to only store a subset of subscriptions and match a subset of publications.

Some properties of the pub/sub model are useful even in small networks. One such advantage is the *decoupling* of publishers and subscribers, which encourages the loose coupling of distributed software components. The pub/sub model fits very nicely into the emerging Service Oriented Architecture [13, 4] (SOA). In a SOA architecture, a distributed application is built using loosely coupled, reusable services, with an Enterprise Service Bus (ESB) providing a communication fabric among the services. Figure 1 illustrates a SOA architecture of a Web server that consists of a frontend application server and two backend databases. The ESB that connects these services is essentially a pub/sub broker network, and the communication between services is message-based, with the bindings among the services specified by subscriptions. That is, messages do not have to be sent to network addresses, but to "virtual" destinations. For example, the products database in Figure 1 would subscribe to product queries, and queries from the application server for product information would not be sent to the IP address of the products database, but simply published to whatever database has expressed (through its subscription) the ability to handle product queries. Notice that the interaction between the services are specified with *declarative bindings*. Subscriptions and publications support a higher level of abstraction when binding services together.

The SOA architecture realized through a pub/sub ESB also supports *incremental service deployment*. For example, it is trivial to add a monitoring service to record all queries from the application server service, by plugging such a service into the ESB, as shown in Figure 2; this service would subscribe to all messages from the application server that is of type queries. Notice that absolutely no changes are required to any existing service. This is possible because the pub/sub ESB performs *late binding* of the declarative bindings between the services. That is, not only are the bindings between services specified with declarative subscriptions, these bindings are evaluated at runtime. As another example of incremental service deployment, consider a products database that can no longer handle an increasing load. The products catalog can be partitioned into two databases—say one for new products, and one for legacy products—and the current products database can be easily removed from the ESB and two new databases connected to the bus. Due to the intelligent routing of the pub/sub ESB, queries from the application server will be routed automatically to the appropriate products database.

3.2 Distributed Hash Table Benefits

Some of the more compelling benefits of DHTs are *organic scaling* and *self-organization*, both of which become more beneficial with increasing network size. Organic scaling, or infrastructure-less scaling, is the property that allows DHT networks to automatically scale with the load on the network: as the load on the network increases (through more peers joining the network), the aggregate resources in the network naturally grows since peers are required to donate their resources. Also, the self-organization of DHT networks removes the complex administrative task of managing large network topologies.

However, just as with pub/sub, DHTs are useful in small networks too. For example, consider a small corporation with limited capital running a distributed application over a DHT substrate. DHTs make *effective use of cheap commodity hardware* due to their fault-tolerance and automatic load balancing properties.

Also, the *incremental scalability* of DHT networks allows the corporation to simply plug in more commodity machines to the DHT as required; there is no need to rewrite the distributed application or reconfigure the network topology. Together, these properties allow the corporation to build an incrementally scalable distributed application using cheap commodity components.

There is a harmonious intersection of the small-scale benefits of pub/sub and DHTs, namely the incremental development and deployment of distributed applications and networks. This, on its own, is a compelling argument for implementing a pub/sub system over a DHT substrate, as described in Section 4.

4 DHT Publish/Subscribe Design

The topic-based pub/sub matching and routing problems have been addressed in P2P networks [3, 20]. However, the techniques used in topic-based P2P pub/sub cannot be trivially extended to content-based pub/sub in P2P networks. The problems become evident when subscriptions with range predicates are used.

Despite the benefits of structured P2P networks, it is non-trivial to build a content-based pub/sub system over a DHT. In particular, a hash table is not well suited for performing range queries. It is typically necessary to "walk" the range to find all matching entries in the hash table. This problem is exacerbated when the data items are continuous (floating point) values. Finally, range queries in a distributed system introduce the issues of data placement and query routing.

4.1 Distributed Multidimensional Matching

We now develop the distributed multidimensional matching (DMM) algorithm, which can match multiple attributes simultaneously. The DMM algorithm maps the pub/sub matching problem to one of multidimensional indexing.

The algorithm assumes each attribute has a known domain (for example, between $[1, 2^{32}]$), a known finest granularity (for example, 10 units), and that there is a known global order of the attributes in the system (for example, the lexicographic ordering of the attribute names).

4.1.1 Mapping Pub/Sub to Multidimensional Indexing

The mapping from the pub/sub domain to a spatial domain is as follows: (1) A d-dimensional space S is created, where d is the number of unique attributes in the pub/sub domain. (2) Every attribute a_i in the pub/sub domain maps to a dimension d_i in S.

A d-dimensional space S is managed by a binary search tree that represents a recursive subdivision of the universe into subspaces (regions) by means of $(d-1)$-dimensional hyperplanes. The hyperplanes are iso-oriented and their direction alternates among the d possibilities, with each hyperplane dividing a region in half. Each region r has a corresponding node $n(r)$ in the search tree.

Each region is addressed by a bit string, called a z-code, and is associated with one node in the tree. Figure 4 presents the algorithm to convert an integer attribute to a z-code. The z-code of a region is computed by interleaving the z-codes of the corresponding ranges of each dimension

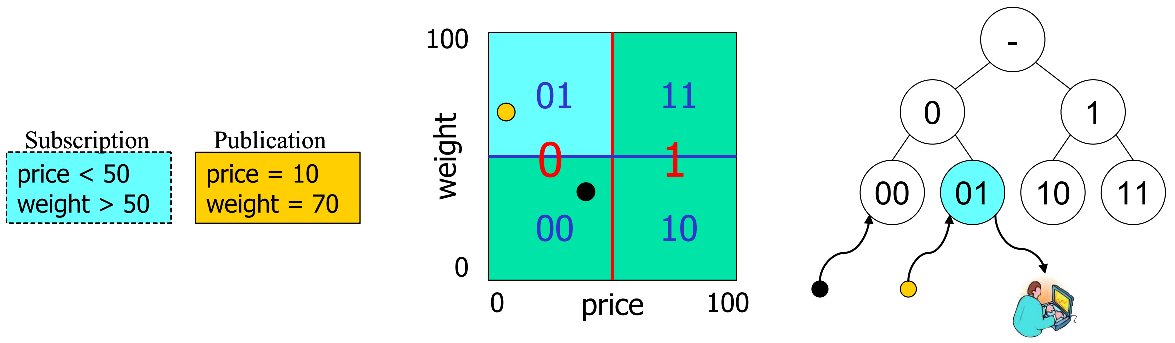


Figure 3. Mapping from pub/sub to spatial to network domain

```
Algorithm IntegerAttributeToZCode($lval, uval, lbnd, ubnd$)
(∗ Attribute has value [lval,uval] and bounds [lbnd,ubnd] ∗)
1.  $l \leftarrow lbnd$
2.  $u \leftarrow ubnd$
3.  $zc \leftarrow$ nil
4.  $lastIter \leftarrow$ false
5.  repeat
6.      $m \leftarrow \frac{l+u}{2}$
7.      if $lval \leq m \wedge uval \leq m$
8.        then $u \leftarrow \lfloor m \rfloor$
9.             $zc \leftarrow zc.Append(0)$
10.       else if $lval > m \wedge uval > m$
11.              then $l \leftarrow \lceil m \rceil$
12.                   $zc \leftarrow zc.Append(1)$
13.              else (∗ Doesn't fit in either half. ∗)
14.                   stop
15.     $lastIter \leftarrow l == u$
16. until $lastIter$
17. return $zc$
```

Figure 4. Determining the z-code

```
Algorithm AttributesToZCode($attrs$)
(∗ Return the z-code of the specified attributes ∗)
1.  (∗ Determine the minimum z-code of all attributes. ∗)
2.  $minlen \leftarrow MinZCodeLength(attrs)$
3.
4.  (∗ Weave zcodes into one. ∗)
5.  $zc \leftarrow$ nil
6.  for $i \leftarrow 0$ to $minlen - 1$
7.      do for $attr \in attrs$
8.             do $z \leftarrow ZCodeOf(attr)$
9.                $bit \leftarrow GetBit(z, i)$
10.               $zc \leftarrow zc.Append(bit)$
11. return $zc$
```

Figure 5. Determining the z-code of a set of attributes

that constitute that region, as shown by the algorithm in Figure 5.

A subscription (object) s is stored at all the leaf nodes $n(r_i)$ in the search tree such that r_i intersects s. Thus the insertion or deletion of a subscription requires the traversal of multiple paths from the root to leaves. An event (point) e will find matching subscriptions by traversing a single path from the root to a leaf, where matching subscriptions will be found.

A leaf node with an excessive number of subscriptions can create two children and move its subscriptions to them. This load balancing technique is referred to as *subscription delegation*. The splitting/merging of regions is done dynamically based on local decisions. If the number of subscriptions stored at a node $n(r)$ exceeds some threshold, then region r is split into r' and r'', and new nodes $n(r')$ and $n(r'')$ are created. The z-code of the new region r' (r'') is the z-code of r with bit 0 (1) appended. A subscription s at node $n(r)$ is sent to $n(r')$ ($n(r'')$) if s intersects r' (r''). Notice that s may be delegated to one or more child nodes depending on how coarse grained s is. This increases subscription load, but allows events to only traverse one path down the tree to find matching subscriptions.

4.1.2 Mapping Multidimensional Indexing to a DHT

Each region r with z-code z has a corresponding node $n(r)$ in the tree. The information of each node is stored at the peer $p(r)$ (as determined by the DHT) in the network. Note that given the z-code of a region r, peers can independently find $p(r)$.

We begin with a single root peer $p(S)$ for the entire space. In order for both publishers and subscribers to find this root, $p(S)$ can be the hash of the attributes in the system (which is known to all peers). Subscriptions are sent to the root peer $p(S)$ and flow down to the appropriate leaf nodes. To avoid the root peer from becoming overloaded, an event e flows up the tree to find matching subscriptions. We can find the smallest region r that encloses e, and send e to $p(r)$. If $n(r)$ doesn't exist in the binary search tree, $p(r)$ forwards e to its parent $p(r')$ in the tree. An example of a complete mapping from the pub/sub to spatial to network domain is shown in Figure 3.

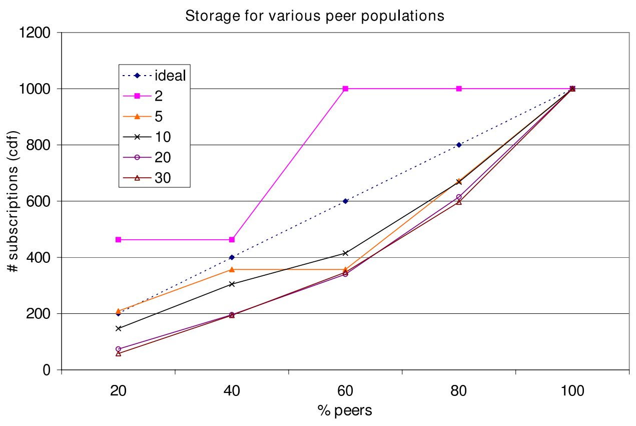


Figure 6. DHT key storage distribution

4.1.3 Algorithm

The propagation of a subscription s goes through two stages. First, during the *finding tree* stage s travels towards the DMM tree, with every peer along the path storing s; the reverse path of these subscriptions builds the multicast tree. Once s has found a node in the tree, it then goes into the *finding leaf* stage. In this stage, it travels up or down the tree searching for an existing leaf node; s is not stored at the peers in this stage until it reaches an existing leaf node. Publication propagation is similar to that of subscriptions.

5 Evaluation

The experiments are run on SimPastry [3], a Pastry simulator, with the DMM algorithm implemented on top of SimPastry.

The main metric studied in this paper is the storage load on the peers in the system. Only the storage of subscriptions in the DMM tree structure are counted. Notably, subscriptions that are used to create the multicast tree are ignored. The number of hops in the multicast tree varies with the size of the network, the efficiency of the DHT routing protocol, and the locations of the subscribers. Therefore in order to keep the results independent of secondary variables, only the DMM structure's storage performance is measured.

Unless otherwise specified, the subscriptions in the workloads consist of a single attribute whose name is selected randomly, and whose value is an integer range with lower and upper bounds in [1,2^{18}]. The number of peers in the DHT is varied from 2 to 30.

5.1 DHT Key Storage

We first evaluate the performance of the underlying DHT substrate without considering the pub/sub protocol. This experiment inserts 1000 randomly generated (key,value) pairs into the DHT and measures the distribution of keys in the network. Figure 6 shows a cumulative distribution function of the storage load among the peers in the DHT for various peer populations. A point (x, y) in the graph means that the x percent of peers with the smallest storage load store y percent of the keys in the system. The ideal distribution is a straight line, and Figure 6 shows that the keys are indeed well balanced among the nodes. Note that the step function-like CDF for the case with 2 peers is an artifact of the fact that there are only two peers but five data points on the horizontal axis.

Although not shown here, the lookup of keys in the DHT averages only one hop due to the aggressive caching of routes by the DHT.

5.2 Fine Grained Subscriptions

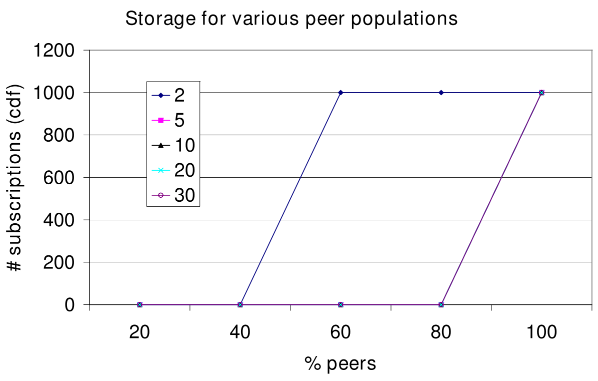
Now we evaluate the performance of our pub/sub design. In this experiment 1000 subscriptions are generated with the same attribute name but random value; the upper value is set equal to the lower value. This results in fine grained subscriptions, that is, subscriptions with very specific interests. Since subscriptions have the same attribute name, subscriptions are initially clustered; this is done to emphasize the effects of the load balancing algorithm.

Figure 7(a) shows a CDF plot of the distribution of subscriptions among the peers in the network. We see that the storage load is highly unbalanced, with a few peers shouldering all the load. However, the results improve when the subscription delegation load balancing feature of the DMM algorithm is enabled. Figure 7(b) shows that subscription delegation results in subscription load that is distributed similar to the DHT keys in Figure 6.

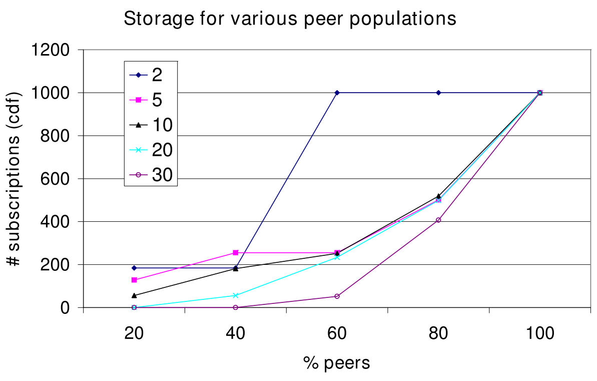
5.3 Coarse Grained Subscriptions

The above results suggest the DMM structure balances subscription storage well, but this is not true in all cases. In this experiment the subscriptions have a random lower and upper value, resulting in coarser grained subscriptions than the previous workload. For this workload, Figure 8(a) again shows that without load balancing enabled, the distribution is highly uneven. When load balancing is enabled the load is again relatively evenly balanced in Figure 8(b).

However, while the shape of the storage load looks balanced in Figure 8(b), an examination of the absolute values leads to a different conclusion. The total load with load balancing enabled is more than twenty times the number of total subscriptions in the system. This is because the subscription delegation algorithm in the DMM structure is not efficient when the subscriptions are coarse grained. Recall that when an overloaded node in the DMM tree wishes to delegate a subscription to its children, it may send the sub-



(a) no load balance



(b) load balance

Figure 7. Fine grained subscriptions

scription to both its children if the subscription is too coarse grained to be enclosed by the region indexed by any one of its children. Therefore, a single coarse grained subscription may result in up to m copies in the system after m delegations.

The explosion of subscriptions illustrated in Figure 8(b) occurs because of an assumption of the DMM algorithm: there are *some* peers in the DHT that are not overloaded. Therefore, after enough delegation steps, the subscriptions will find their way to and terminate at a peer with sufficient resources. However this assumption is less likely to hold in small networks, where all the peers may become overloaded. Subscription delegation (for coarse grained subscriptions) makes the situation worse by increasing the number of subscriptions and storing them on already overloaded peers. In fact, delegation will continue recursively until a predefined limit (which is related to the known finest granularity of an attribute's domain) is reached.

To solve runaway subscription delegation requires that a node has knowledge about the unused resources in the DHT, so that delegation can stop when there are no nodes with some threshold of unused resources. Acquiring such global knowledge is usually not practical in distributed systems, but may be estimated. For example, each peer may probe the available resources of some random set of peers and assume that these few peers are representative of the entire network. We plan to pursue such techniques in future work.

The load balancing technique that assumes there are sufficient unused resources (whether storage, bandwidth, or processing cycles) somewhere in the network occurs in several DHT applications, such as the active caching and replication techniques in the CAN DHT [19], and the routing hotspot circumvention in the Meghdoot [8] pub/sub design. This assumption is often implicit and is rarely given a second thought. However, as we have argued above, this assumption may not be true in small-scale networks.

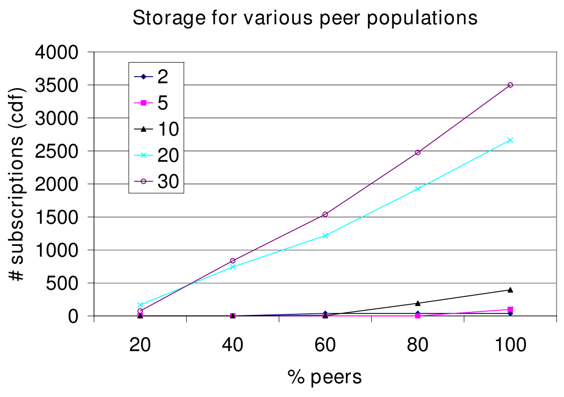
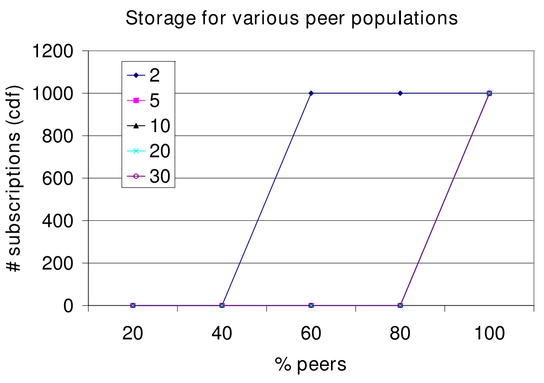


Figure 9. Coarse grained subscriptions (covering and load balance)

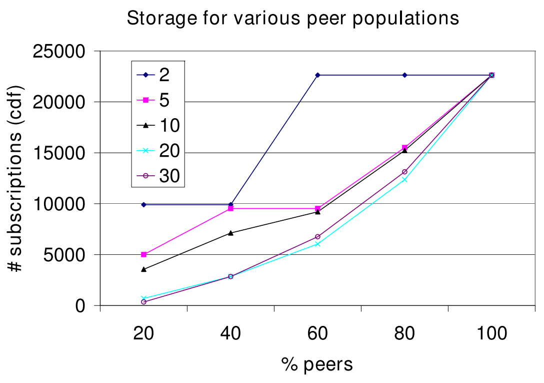
5.4 Subscription Covering

The storage of coarse grained subscriptions can be improved with the covering [2] optimization. Covering quenches subscriptions before they reach the DMM tree. A subscriptions s covers subscription s' if the set of events E that match s is a superset of the set of events E' that match s'. That is, a more general (or coarse grained) subscription covers a more specific (or fine grained) one. If the paths of s and s' from their respective subscribers to the DMM tree intersect at peer p , then p only needs to forward s.

Figure 9 shows the storage distribution with load balancing and covering enabled for coarse grained subscriptions. We see that the total number of subscriptions is greatly reduced from Figure 8(b) where covering was disabled. Fur-



(a) no load balance



(b) load balance

Figure 8. Coarse grained subscriptions

thermore, the load decreases as the number of peers in the system decreases. This is a nice property, since the total load increases only when more peers are added to the network; the aggregate load grows as the aggregate available resources increases. It is important to point out, however, that this property is not due to the load balancing algorithm itself, but is a side effect of the covering optimization: the paths of any two subscriptions are more likely to intersect, and hence be covered, where there are fewer peers in the network.

6 Related work

The closest work related to this paper is Service Oriented Architectures [13, 4] (SOA). The Enterprise Service Bus (ESB), which plays an integral role in the SOA model is often a simplified pub/sub or messaging broker network. However current ESBs are proprietary systems [5, 14, 9] and few implementations or performance details are available. We are not aware of any academic research into small-scale pub/sub or DHT environments. In order to provide some context for this work, this section discusses related designs that implement a pub/sub system over a DHT.

Scribe [3] is a *channel-based* pub/sub system built over the Pastry DHT. Scribe treats a channel name c as a key in the DHT which is stored at peer r called the channel root. Subscriptions are sent towards r, and their reverse path builds a multicast tree from the channel root r to the subscribers. Publications are also sent to the channel root, and then follow the multicast tree to the subscribers in the channel.

Hermes [18] is a content-based pub/sub system built over the Pastry DHT. It essentially assigns a channel to each publication and subscription and the matching algorithm degenerates to that of Scribe. Unlike the algorithm in this paper, Hermes' initial matching algorithm does not discriminate based on content, resulting in unnecessary load on the channel root peer.

Meghdoot [8] is a content-based pub/sub system built over the CAN DHT. For an application with k attributes, Meghdoot constructs a CAN space of dimension $2k$. Subscriptions are mapped to a point in the CAN space and stored at the responsible node. Publications traverse all regions with possible matching subscriptions. Meghdoot handles routing load by splitting a subscription at a peer to its neighbors. As discussed in Section 5, this may lead to an explosion of subscriptions in small-scale networks where all nodes are already overloaded.

This work—referred to as P2P-ToPSS [22]—is part of the ToPSS (Toronto Publish/Subscribe System) research projects [10, 12, 11, 15, 16, 17].

7 Conclusion

Distributed pub/sub systems can benefit from P2P networks. Notably, scalability is possible without dedicated infrastructure. The Pastry DHT is a P2P overlay that provides probabilistic performance guarantees. We develop an algorithm that implements a pub/sub system over a DHT.

Research into pub/sub systems and DHT networks have traditionally focused on their large-scale benefits. However, both pub/sub systems and DHTs have properties that benefit small-scale networks as well, namely the decoupling of producers and consumers, and incremental service deployment inherent in the pub/sub model, and the effective use of commodity hardware, and incremental scalability abilities of DHTs.

Experiments show that in small networks (with less than 30 peers) DHTs continue to exhibit good storage load balance of (key,value) pairs, and lookup costs. The former is because the DHT hash function distributes keys evenly throughout the identifier circle, and the latter is due to aggressive route caching in the Pastry DHT. Good load balance is also achieved by our DMM pub/sub indexing algorithm under certain workloads, particularly, fine grained subscriptions. However, coarse grained subscriptions are not indexed efficiently by the DMM structure, and cause an explosion in the total subscription load.

The large increase in subscription load is due to a fundamental assumption in DHT applications that unused resources are available somewhere in the network. This assumption is necessary because there is no global knowledge of unused resources in a large DHT network. In small networks, however, the assumption is not valid, as it is conceivable that every peer is already overloaded. The important point illustrated by this work is that data management applications with proven scalability in large P2P networks may not scale in small networks. This arises from the flawed assumption of ample aggregate available resources.

For future work, we would like to evaluate more metrics. In particular, some preliminary results indicate that while the aggregate bandwidth requirements to perform pub/sub matching decreases as the network gets smaller, the average load on each peer increases. We would like to investigate this further. We also plan to implement some heuristics to stop the runaway subscription delegation in cases where doing so will not improve the system-wide load balance; that is, we would no longer assume there are unused resources in the network. Another area for future research is to obtain real-world workloads to gain a more realistic understanding of the system's performance.

References

[1] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *ICDCS*, 1999.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM ToCS*, 19(3):332–383, Aug. 2001.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), oct 2002.

[4] K. Channabasavaiah, K. Holley, and J. Edward M. Tuggle. Migrating to a service-oriented architecture. http://ibm.com/developerworks/webservices/library/ws-migratesoa/, 2003.

[5] D. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[7] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD*, 2001.

[8] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: Content-based publish/subscribe over P2P networks. In *Middleware*, pages 254–273, 2004.

[9] F. Ibarra. The enterprise service bus: Building enterprise SOA. http://dev2dev.bea.com/pub/a/2004/12/soa_ibarra.html, Dec 2004.

[10] G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *ICDCS*, Columbus, Ohio, 2005.

[11] H. Liu and H.-A. Jacobsen. Modeling Uncertainties in Publish/Subscribe System. In *ICDE*, 2004.

[12] V. Muthusamy, M. Petrovic, and H.-A. Jacobsen. Effects of routing computations in content-based routing networks with mobile data sources. In *MOBICOM*, Cologne, Germany, August 2005.

[13] Y. V. Natis. Service-oriented architecture scenario. http://www.gartner.com/DisplayDocument?doc_cd=114358, Apr 2003.

[14] C. Nott, P. Edwards, A. Humphreys, and M. Keen. Using message sets in websphere business integration message broker to implement an ESB in an SOA. http://www.redbooks.ibm.com/abstracts/redp3978.html, 2005.

[15] M. Petrovic, I. Burcea, and H.-A. Jacobsen. S-ToPSS – a semantic publish/subscribe system. In *VLDB*, Berlin, Germany, September 2003.

[16] M. Petrovic, H. Liu, and H.-A. Jacobsen. G-ToPSS – fast filtering of graph-based metadata. In *WWW*, Chiba, Japan, May 2005.

[17] M. Petrovic, V. Muthusam, and H.-A. Jacobsen. Content-based routing in mobile ad hoc networks. In *MobiQuitous*, July 2005.

[18] P. R. Pietzuch and J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *DEBS Workshop at SIGMOD/PODS*, pages 1–8. ACM Press, 2003.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[20] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *NGC Workshop at SIGCOMM*, pages 14–29, 2001.

[21] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *ICDSP (Middleware)*, pages 329–350, Nov. 2001.

[22] D. Tam, R. Azimi, and H.-A. Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. In *DBISP2P Workshop at VLDB*, September 2003.

[23] Y. Zhao and R. Strom. Exploiting event stream interpretation in publish-subscribe systems. In *PODC*, pages 219–228. ACM Press, 2001.