State Aware Data Dissemination over Structured Overlays

Liping Chen Gul Agha

University of Illinois at Urbana-Champaign
201 N Goodwin Ave, Urbana, IL 61801 USA
E-mail: {lchen2,agha}@uiuc.edu

Abstract

We describe the problem of data dissemination in stream-oriented applications where the required filter is a function of the current state. We call such functions dynamic filters. *A State Aware Data Dissemination Network (SA-DDN) is proposed to support dynamic filters. Two approaches Single-level Filtering (SF) and Multi-level Filter Decomposition (MFD) are proposed to facilitate the data dissemination. We show how MFD improves performance over SF. We then describe a realization of SA-DDN on top of an improved bi-directional Chord overlay with a built-in multicast mechanism. An application of stock price monitoring is implemented based on SA-DDN and real life stock quotes are collected to demonstrate the feasibility of our system. Extensive simulations are performed to compare the performance of both approaches and provide insight into the advantages of MFD.*

1. Introduction

The advancement of the P2P paradigm has proved it to be a feasible means for data dissemination and file storage. Recently, there has been an increased interest in applying P2P techniques to publish/subscribe systems [3, 17, 5, 7, 12, 1]. One common characteristic shared by all the above applications is that the data filter is static: once a subscriber submits a filter, the filter no longer changes. In this work, however, we consider applications where subscribers register consistency requirements to the Data Dissemination Network (DDN). The consistency value indicates the maximum inconsistency allowed between the data source and destinations. We call this type of filters *dynamic filters*: the value of the filter varies with the current state of individual subscribers. We use this setting to study applications where the consistency of data could be traded for communication overhead.

In the case of streaming data, such as stock quotes or sensor readings, we see an opportunity for a new pattern of data dissemination. Observe that with the data that is generated continuously in large quantities, it is infeasible to deliver every piece of datum generated. Fortunately, for such data sets, subscribers are not interested in exact values of the data stream, but in a quantitative approximation to these values. For example, in a distributed sensor network that continuously monitors environmental conditions such as light and temperature, a bounded approximation of sensor readings could greatly reduce the communication overhead and hence prolong the lifespan of sensors' batteries. Other targeted applications may include stock quotes services, sports score monitoring, wide-area resource accounting, etc.

Dynamic filters are filters moving with the current state of each subscriber and hence a naive way to support dynamic filters in Pub/Sub systems with static filters is to continuously de-register and re-register the filters as the underlying state changes. Such repeated registration incurs a high publication cost. The obvious way to reduce such publication cost is to store the filter in terms of the consistency requirement together with the local states last communicated to each subscriber. We call this scheme *Single-level Filtering* (SF). We improve on SF by proposing *Multi-level Filter Decomposition* (MFD). MFD decomposes the filters in such a way that proper structures could be constructed to prune the dissemination.

The main contributions of our work are twofold. First, we recognize the importance of supporting dynamic filters in Pub/Sub systems. To the best of our knowledge, this paper is the first to study dynamic filters over structured overlays. Second, we describe two approaches, namely SF and MFD, to address the problem. Comprehensive experiments are conducted to compare the performance and demonstrate the feasibility of our approaches in medium-sized networks. We implement our system using bi-directional Chord with a built-in multicast mechanism.

It is important to point out some limitations of our approach. First, the approach is targeted for quantitative data streams only. Qualitative data streams such as news feed, blog entries would need to be dealt with differently. Second, MFD in particular, only works with symmetric consistency bounds (i.e. where the consistency value defines half the consistency range).

The rest of the paper is organized as follows. Section 2 discusses some of the related work. Section 3, formally defines dynamic filters. Section 4 presents the overall system design and various aspects of building the system. Section 5 shows our evaluation methodology and results. Section 6 concludes the paper with a brief discussion and future work.

2. Related Work

Recent years have seen an increased interest in building Pub/Sub systems on top of structured P2P overlays. Much of the previous work has leveraged the self organization, scalability and fault tolerance capabilities of structured overlays to build Data Dissemination Networks (DDNs). Scribe [3] and Bayeux [17] are two earlier attempts to build subject-based pub/sub systems on top of Pastry[9] and Tapestry [16] respectively. More recent work focuses on content-based pub/sub systems, which support more expressive subscriptions of finer granularity and possibly range constraints. These approaches differ in the choice of underlying overlays and how subscriptions are mapped to overlay addresses. In particular, [5] supports subscriptions of range constraints by converting a n-dimensional range to a point in 2n-dimensional space and deriving a proper scheme to map logical partitions to CAN [8] nodes. [7] builds over Pastry [9]. By choosing the right combination from foward and reverse path of subscription and advertisement, it supports both subject-based and content-based subscriptions. Both [12] and [1] use Chord [11] as the underlying overlay. [12] proposes an order preserving Chord where each attribute domain occupies a continuous segment of the Chord ring, while [1] introduces a general architecture that adopts an abstract stateless mapping.

All these previous works deal with *static filters*. Although dynamic filters or equivalent ideas are emerging as a new pattern in data dissemination and collection, to the best of our knowledge, no significant works have been dedicated to study dynamic filters over structured overlays. We consider [10] as most related to our work. [10] defines the notion of coherence, which is equivalent to the notion of consistency in our work. However, [10] builds on unstructured overlays , where the data assignment and the client assignment problem are considered separately. With the help of structured P2P overlays, we could address both problems using a unified framework with performance guarantee.

We observe the counterparts of dynamic filters in other research contexts, with different concerns and solutions. [6] addresses the problem of persistent queries with precision requirements over continuous data streams. However, their approach is centralized, which does not scale to large networks. In the context of replicated network service, [15] focuses on algorithms to efficiently bound absolute error among replicated services using only local information. However, they are not concerned with data dissemination or other network issues such as self organization, scalability. [2] and [14] have studied consistency maintenance in the context of web caching. In web caching, the staleness of the data object is measured by time while in our context, it is measured by value. Maintaining consistency based on time is considered as a simpler problem since time only monotonically increases while values could fluctuate in both directions.

3. Problem Statement

A typical content based subscription is usually represented in a range form: $d \in [2, 5]$, suppose d represents the data item that the subscriber is interested in. Once the filter is registered into the network, it remains the same throughout its lifetime. In a SA-DDN, however, situations change. A subscriber s specifies a maximum consistency value c_{sd} for d, which can be translated to a content based subscription as $[\Theta_d^s(t) - c_{sd}, \Theta_d^s(t) + c_{sd}]$, where $\Theta_d^s(t)$ denotes the state of subscriber s for data source d at time t. Only data values that fall outside this range are to be delivered to the subscriber s. After each content delivery, the state of the subscriber is refreshed and the filter is recentered. Hence in SA-DDN, the filter moves with the state of the subscribers.

To formally define the problem, we let $\mathbb{D}$ denote the set of all data items and let d range over $\mathbb{D}$. Each data item d represents a data source which continuously generates a data stream v_d and $v_d(t)$ returns the value at time t. We use $\mathbb{S}$ to denote the set of subscribers and let s range over $\mathbb{S}$. One data item might be interesting to many subscribers and one subscriber might be interested in many data items. We use $\mathbb{D}_s$ to represent the subset of data items that s is interested in.

A subscription is a vector $(subID, d, c_{sd}, s)$. To guarantee the uniqueness of each $subID$, we keep a counter at individual subscriber. The $subID$ is constructed by concatenating the subscriber s with the sequence number. The consistency value c_{sd} indicates how much inconsistency that the subscriber is willing to tolerate between the value at the data source and locally.

For each data item $d \in \mathbb{D}_s$, subscriber s keeps a sequence of local states Θ_d^s which is a subset of v_d. In an ideal situation where there is no message delay, this inequality holds for any time instance t: $|v_d(t) - \Theta_d^s(t)| < c_{sd}$. Based on the local state $\Theta_d^s(t)$ and the consistency value c_{sd}, we can compute the consistency range $\zeta_d^s(t)$ as $[\Theta_d^s(t) - c_{sd}, \Theta_d^s(t) + c_{sd}]$. The consistency range defines a safe zone for the subscriber centered at $\Theta_d^s(t)$. c_{sd} defines half of the range width.

Some basic assumptions are made which are commonly adopted by most of today's DDN when building our system. First, each subscriber or publisher knows at least one broker node, the *bootstrap* node. Second,

we assume unique data source or multiple data sources with *no* data conflicts. The problem to integrate conflicting data from multiple sources is in itself an open research problem and hence orthogonal to the problem that we are to address here. Third, without loss of generality, data values are converted to integers. Individual data items and hence the consistency values are bounded, although may be of different range.

4. System Design

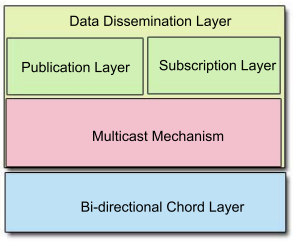


Figure 1. The System Design of the DDN

Figure 1 shows the block design diagram of SA-DDN. The underlying layer is the bi-directional chord routing layer. We believe that multicast is an intrinsic mechanism for our problem. Since multicast is not part of the Chord protocol, we build another multicast mechanism on top of the Chord layer. The main part of our system is the DDN layer, which consists of two functional layers: the publication and the subscription layer. In the rest of this section, we will discuss each layer in detail.

4.1. Bi-directional Chord Layer

Chord [11] is one of the most influential overlay in the realm of Distributed Hash Tables (DHTs). Routing in original Chord is uni-directional, which is not optimal. [4] discussed optimal routing in Chord which routes in both directions along the circular ring. For base 2, the optimal average path length is $O(b/3)$ where b is the diameter of the ring. However, in order to realize such algorithm, we need to double the size of the finger table. To keep the size of original finger table, instead of base 2, we use base 4 and create fingers for both directions. For either direction, the ith entry stores information about those nodes which is at least 4^{i-1} apart from the current node. Once we get the entries for both directions, the final finger table is constructed by sorting all the fingers from both directions and remove the duplicates. Figure 2 shows the detailed greedy bi-directional routing algorithm. ClosestFinger(k) returns the finger closest to k in terms of absolute distance. The absolute distance is computed by taking the minimum of distances computed in both clockwise and anti-clockwise directions. The original Chord uses only clockwise distance.

Figure 3 shows the comparison of uni-directional chord and our bi-directional chord in terms of average path length. We also show the performance of optimal routing for comparison purpose. The number of nodes increases from 100 to 3200, doubling each run. Our bi-directional Chord gives a performance which is close to optimal.

```
n.FindSucc*(k) :
    if k ∈ (n, n.successor]
        return n.successor;
    else if k ∈ (n.predecessor, n]
        return n;
    else
        return n.ClosestFinger(k);
```

Figure 2. The algorithm for finding the successor of key k on each node

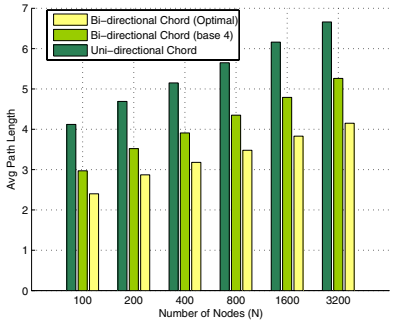


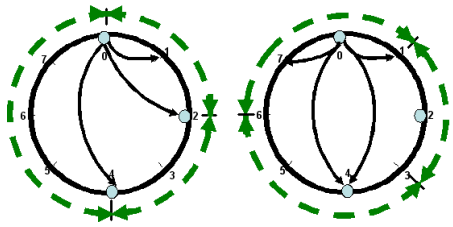
Figure 3. Comparison of bi-directional and uni-direction Chord in terms of average path length

4.2. Multicast Layer

As Chord is mainly designed as a point-to-point protocol, it lacks a basic multicast mechanism which is indispensable in a data dissemination network. Since in Chord a linear order is imposed on all the participating nodes to form the ring, this characteristic could be used to construct an efficient multicast primitive.

If we view *broadcast* in Chord as forming a spanning tree rooted at the node which initiates the action, multicast in Chord effectively prunes the spanning tree to explore only those branches which lead to required destinations. [1] implements a multicast primitive with uni-directional Chord. We now show how we implement a generic multicast primitive based on bi-directional Chord. Given a list of destinations and a sorted finger table, each finger node is only responsible for forwarding messages to those destinations whose *ChordID* is the closest. Hence if in multicast with uni-directional Chord, the responsibility range of each finger $f[i]$ is $(f[i], f[i+1]]$, in multicast with bi-directional Chord, the responsibility range of finger $f[i]$ changes to $(f[i-1] + (f[i] - f[i-1])/2, f[i] + (f[i+1] - f[i])/2]$. Figure 4 demonstrates such difference. The arrowed lines show the responsibility range. Figure 5 gives the detailed multicast algorithm. For illustration sim-

plicity, we treat current node as $f[0]$ in finger table. PROCESS(d) is a generic function indicating the processing of data once they reach the destination.



a) Uni-directional Chord b) Bi-directional Chord

Figure 4. Comparison of responsibility range for uni-directional and bi-directional Chord

```
MULTICAST{data, dlist} :
    d = EXTRACTDESTS(dlist, n.predecessor, n);
    if d ≠ ∅
        dlist = dlist \ d;
        n.PROCESS(d);
    d = EXTRACTDESTS(dlist, n, n.successor);
    if d ≠ ∅
        dlist = dlist \ d;
        send MULTICAST{data, d} to n.successor;
    for i = 1 to m − 1
        d = EXTRACTDESTS(dlist,
            n.f[i − 1] + (n.f[i] − n.f[i − 1])/2,
            n.f[i] + (n.f[(i + 1) mod 2m] − n.f[i])/2);
        if d ≠ ∅
            send MULTICAST{data, d} to n.f[i];
```

```
n.EXTRACTDESTS(dlist, n1, n2) :
    return {k ∈ dlist | k ∈ (n1, n2]};
```

Figure 5. Multicast algorithm on each node

4.3. Data Dissemination Layer

Since publishers and subscribers are distributed across the network, efficient data delivery usually requires a form of directory service. The directory service of SA-DDN differs from that of general DDN in that they keep not only the subscriptions, their targeted locations, but also the current state of each subscriber.

A centralized directory service was used in earlier DDNs. However, as the DDN expands to much larger scale, a centralized solution is no longer feasible. The directory has to be partitioned and stored at individual nodes. Requirements for doing such partition and storage are twofold: first, both publishers and subscribers should be able to locate the relevant directory service easily without global knowledge; no single node should become a performance bottleneck. With these requirements in mind, we propose two approaches to organize the directory information. MFD is an improvement over SF. Both approaches share the same hashing scheme, which maps whatever logical structure they have to a number of nodes falling within a continuous segment of the Chord ring. In the rest of this section, we will discuss the common hashing scheme adopted by both approaches first and then describe the two different approaches in detail.

4.3.1. Stateless Ring Segment Mapping Scheme We define the hashing function as: $h(d, \delta) = h_1(d) + h_2(\delta)$, where d is the data item and δ is some numeric value pertaining to d. This hash function consists of two portions. We use SHA-1 as h_1 to map data item d to a number k in Chord space, which marks the beginning of the ring segment that d will occupy. The effect of h_1 is to randomly distribute individual data items within the Chord space to achieve load balancing. Depending on the δ value, h_2 determines which node within the ring segment is used to store the information. The range of h_2 is bounded by L, the length of the ring segment. We use a simple linear function as h_2 where $h_2(\delta) = a\delta$ and $a = L/max(\delta)$.

4.3.2. Single-level Filtering (SF) Our Single-level Filtering(SF) approach is straightforward. We simply replace δ by consistency value c in hashing function h introduced in Section 4.3.1. Intuitively, higher consistency value means more tolerance to the inconsistency between source and the local value. Hence if a newly published value doesn't pass through the filter of low consistency value, it shouldn't pass through that of higher consistency value. If this intuition holds, the order preserving property of function h_2 could help us in pruning the ring segment. However, Figure 6 shows that this might not be the case.

Data stream:	10 16 21 25 30 25 19 13 8 $\cdots$
Delivered to $s_1(c_1 = 5)$:	10 16 25 19 13 $\cdots$
Delivered to $s_2(c_2 = 10)$:	10 21 8 $\cdots$

Figure 6. Example to show data delivered to subscribers with different consistency requirements

In Figure 6, the first row shows the time series data generated at the source. The rest rows show the actual data values delivered to subscriber s_1 and s_2, with consistency values 5 and 10 respectively. Note that although value 8 does not pass through the filter of 5, it passes through the filter of 10, in contradiction to our intuition.

An implication from this observation is: if we treat consistency values as single-level filters without decomposition, a structured organization of filters most probably won't help to prune the search space. This is why a newly published value $v_d(t)$ has to be sent to the entire ring segment rather than to only a portion of it.

The subscription layer is straightforward for SF. Given a subscription $(subID, d, c_{sd}, s)$, we use $h(d, c_{sd})$ to decide the target node for storing the subscription.

The subscriber is assigned an initial state for data item d based on the most recent value of d at the source.

The publication layer is more involved. Each time data source d refreshes its data, this new value is multicast to the entire ring segment. The underlying multicast mechanism shown in Figure 5 could be used with only minor modifications: instead of accepting a set of destinations, it accepts a destination range. The resulting message is MCAST2SEG$\{data, R\}$, where $R = [k, k + L)$, k is the beginning and L the length of the destination range. Replacing all the sets with ranges and set operations with range operations, we derive the algorithm for MCAST2SEG$\{data, R\}$.

Once the newly published data value is delivered to the ring segment, it is checked against the directory information stored at each node. The data value is delivered to those out-of-sync subscribers using the underlying multicast layer.

4.3.3. Multi-level Filter Decomposition (MFD) SF shows that if we treat each consistency value as a single-level filter, it is hard if not impossible to construct a structure to facilitate pruning of the search space. In this section, we attempt to derive a scheme where each filter can be decomposed into a hierarchy of sub-filters, and these sub-filters function together to achieve the result which satisfies our consistency requirements, but usually not optimal.

We still use the data stream presented in Figure 6. Suppose we want to achieve a consistency value of 10. We may decompose it as $4 + 6$, a two-level hierarchy of sub-filters 4 and 6. Figure 7 shows the result.

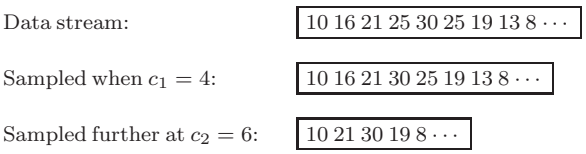


Figure 7. An example of a two-level filter

The main feature that this hierarchy of filters possess is that the output of the lower level is the input of the higher level, hence only those values that pass through the lower level filter will be delivered to the higher level. This feature makes it feasible to build a structure which prunes some part of the ring segment during data dissemination, an improvement over SF.

Theorem 1. *The correctness of the decomposition $C_n \equiv \sum_{i=1}^{n} c_i$, where n is the number of sub-filters.*

Proof: Proof by induction.

The initial case when n = 1 is trivially true.

Assume at level $n-1$, we achieve a filter $C_{n-1} \equiv \sum_{i=1}^{n-1} c_i$. We want to prove that at level n, we achieve a filter $C_n \equiv C_{n-1} + c_n$.

$$C_{n-1} \begin{cases} c_1 & \cdots \\ \vdots & \\ c_{n-1} & s_0, \cdots s_{t-1}, s_t \cdots \end{cases}$$

$$c_n \quad s_0, s_t \cdots$$

Since only the output of level $n-1$ is passed onto level n, the states at level n is a subset of states at level $n-1$. Suppose s_0 is the initial state, among the states $s_1 \cdots s_t$ passing through level $n-1$, only s_t passes through level n. Hence the following inequalities hold:

$$\begin{cases} |s_t - s_{t-1}| > C_{n-1} \\ |s_{t-1} - s_0| < c_n \end{cases}$$

If we solve these two inequalities, we have: $|s_t - s_0| > C_{n-1} + c_n$. Since the current state only depends on the previous state, we could treat s_t as s_0 and continue the same argument for the rest of the states at level $n-1$. Hence we achieve a filter $C_n \equiv C_{n-1} + c_n$ at level n. □

In the previous proof, some other inequalities hold:

$$\begin{cases} |s_i - s_0| > c_{n-1} \\ |s_i - s_0| < c_n \end{cases} \text{ for all } i \in [1, t-1]$$

Since the LHS of both inequalities are the same, in order for these inequalities to hold, we require $c_n > c_{n-1}$. This leads to our Corollary 1.

Corollary 1. *The decomposition $C = \sum_i c_i$ requires that $c_{i+1} > c_i$ for all i.*

Various schemes could be designed as long as it satisfies Corollary 1. In the rest of this section, we will describe the scheme that we adopt.

Logical Structure As discussed, we need to derive a multi-level structure to fit in all consistency values in $[1, max(c)]$ that satisfy the decomposition described in Corollary 1. One easy way of decomposition is to represent the consistency value as a number in base 4. The reason why base 4 is chosen is that it strives for a balance between the number of logical nodes at each level and the number of levels generated. Based on the base 4 encoding, we could easily decide the decomposition of any consistency value. For instance, $21_{(10)} = 111_{(4)} = 001 + 010 + 100_{(4)}$, $34_{(10)} = 202_{(4)} = 002 + 200_{(4)}$.

Based on the encoding, we could decide which node within the logical structure that this consistency value c should be stored. First, the number of non-0s within the encoding decides the number of composing sub-filters and hence the level at which the subscription with c is stored. For instance, a c value of $21_{(10)} = 111_{(4)}$ should be stored at level 3 while $34_{(10)} = 202_{(4)}$ stored at level 2. After deciding on the level, the leading non-0 digit decides on which node at that level c resides. Figure 8 shows a structure of three levels to store subscriptions with maximum consistency value of 63 (inclusive).

Mapping Scheme The mapping is straightforward. A linear order is imposed on the nodes of multi-level structure from left to right and lower level to higher

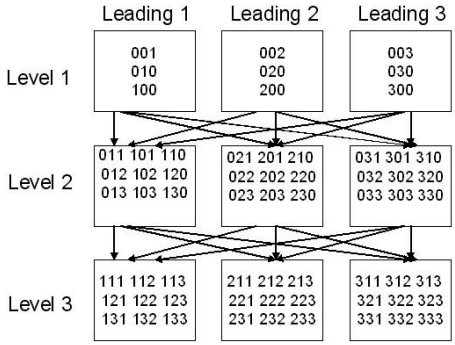


Figure 8. The logical multi-level filter structure

level, starting from 0. We call this the *index* of the logical node. Given the index number ι, we apply $h(d, \iota) = h_1(d) + h_2(\iota)$, where $h_2(\iota) = a\iota$ and $a = L/max(\iota)$. The Chord node whose ChordID immediately succeeds $h(d, \iota)$ stores the logical node ι. It's possible that several logical nodes are mapped to a single physical node.

Subscription Given a subscription $q(subID, d, c_{sd}, s)$, we convert c_{sd} to base 4 encoding. Based on the encoding, we determine the index ι of the logical node and use $h(d, \iota)$ to decide the target physical node for storing the subscription.

However, as opposed to SF, the registration of a single subscription may involve registration at multiple logical nodes at different levels. This is due to the fact that in MFD, the filters at higher level can only be achieved by passing through a path from level 1. For instance, in order to establish a filter of consistency value $21_{(10)} = 111_{(4)}$, we need to establish filters of $011_{(4)}$ and $001_{(4)}$ if they do not already exist. We call these filters *proxy filters*. We represent proxy filters as $\rho(d, c)$. Note that this representation shows that although multiple subscriptions to data item d with the same consistency value c may coexist, only one proxy filter exist for a (d, c) pair. The sole purpose of the proxy filters is to achieve filters at higher levels. Hence proxy filters should be replaced by real subscriptions with $(*, d, c, *)$ if such subscriptions are issued later.

Publication The logical structure of MFD is built in such a way that the output of nodes at level l should be sent to all nodes at level $l + 1$. To allow maximum parallelism, the output of level l should be multicast to $l + 1$ and each node at level $l + 1$ can process the input simultaneously. However, we do not want to create too many messages by allowing each of the node at level l to send messages to level $l + 1$. Hence we use the last node at each level as a leader to compile the output from all nodes at the same level, and only the compiled result is multicast to all nodes at the next level. If no output is generated at level l, no further forwarding is necessary.

The mapping schemes that we adopt guarantee that the logical nodes at the same level will be mapped to a continuous ring segment. Hence the multicast can be easily achieved using MCAST2SEG$\{data, R\}$, where $R = [k, k + l)$ discussed in Section 4.3.2 by changing k for each level and fixing $l = L/max(level)$.

5. Performance Evaluation

In order to evaluate performance of our proposed system architecture in a psudo-reality setting, we build a stock monitoring application based on SA-DDN. The stock price data were provided by [13], which are real life stock quotes collected from Yahoo Finance. The original dataset consists of 30 different stocks, collected in 1 minute interval from Nov. 11th, 2002 to Sep. 12th, 2003. The total data size is about $108MB$. We tailored the dataset such that each stock within a single day are treated as a separate data source. Hence we created a large pool of 5453 data sources. Since how fast the time series data change over time affects the performance of SA-DDN, we classify these generated data sources into 10 groups based on the standard deviation. During our evaluation, data sources are picked from these groups. Two types of distributions are generally more interesting than others: the uniform distribution, where every group has an equal chance of getting picked, and the power law distributions, where certain preferences are exhibited by the users, which makes some groups much more frequently picked than others. We use Pareto distribution in this category.

In order to model the subscriptions, we need to consider the following aspects: the rate of subscription issuance and the distribution of consistency value. We model the arrival of subscriptions as a poisson process; the inter-arrival time is exponentially distributed, thus resulting in highly dynamic network traffic. Again, two types of distributions are used to model the distribution of consistency value, uniform and Pareto.

We evaluate the system performance of SA-DDN using the following metrics: the average cost in terms of subscription cost and publication cost; the average update delay from the time data is published to the time data is delivered to individual subscribers if such delivery is necessary. The subscription cost includes not only the registration cost, but also the cost to maintain the proper structure to facilitate data delivery. The publication cost includes the cost to publish the data to the network and deliver data to the individual subscribers. The average update delay is an indication of how closely local states conform to the data sources. All these metrics are measured in terms of messages exchanged (or hop counts). The reason time is not used is that it heavily depends on hop delays, and irregular hop delays caused by irrelevant factors may bias our evaluation.

5.1. Impact of Different Distributions

In this set of experiments, we would like to study the impact of different consistency value and data distributions on the system performance. Each uses two

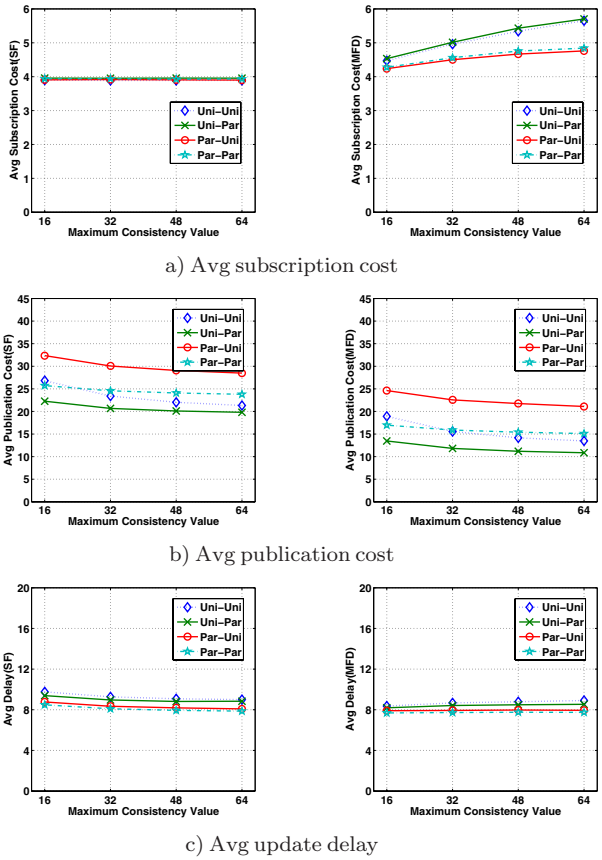
distributions: uniform and Pareto. Hence we have four combinations: Uni-Uni, Uni-Par, Par-Uni and Par-Par. The first term indicates the distribution of consistency value and the second the distribution of data source selection. The maximum consistency value varies from 16 to 64, with 16 incremental. We fix the network configuration with 400 SA-DDN nodes, 200 data sources and 200 subscribers. Each data source generates data at 1 minute intervals, and subscribers generate new subscriptions with mean inter-arrival time of 10 minutes. The simulation length is 4 hours.

Figure 9 shows the performance of both SF and MFD each with 4 different combinations. We leave the comparison between SF and MFD to Section 5.2 since all four combinations exhibit the same trend and focus on comparisons among 4 combinations within each approach. Both SF and MFD do not exhibit significant difference for average subscription cost and update delay as shown in Figures 9a and 9c. However, the average publication costs in Figure 9b vary significantly. For both SF and MFD, the combination Par-Uni gives the highest publication cost, while Uni-Par gives the lowest. To understand this result, we need to understand what these different distributions imply in our context. A Pareto distribution of consistency value implies preference on smaller consistency value, hence more subscriptions are sensitive to even a small data value change. However, a Pareto distribution of data source means a preference in selecting low variation data streams, which implies less local updates to subscribers. Therefore, a combination of Par-Uni necessitates the largest number of updates and hence causes the highest publication cost. In contrast, Uni-Par causes the lowest publication cost.

There is another message conveyed by this set of experiments: a higher publication cost does not necessarily imply a higher update delay. The reason behind this deserves some discussion. The update delay measures the gap between the time data is published and the time data reaches the subscriber, while the publication cost is the cost to deliver the newly refreshed data to all interested subscribers. Hence if we are only concerned with reducing the update delay, broadcast might not be a bad choice since the maximum delay for individual nodes is the diameter of the network. However, the publication cost is prohibitively high. Parallelism might help to reduce individual update delay. To reduce the publication cost, however, we may want to combine as many messages as possible, which may inhibit the parallelism.

5.2. Impact of Network Size

In this set of experiments, we fix the distribution of data source selection and consistency value and study the impact of network size on the system performance. We increase the SA-DDN nodes from 100 to 1600, doubling each time. Since the Par-Uni combination causes



a) Avg subscription cost

b) Avg publication cost

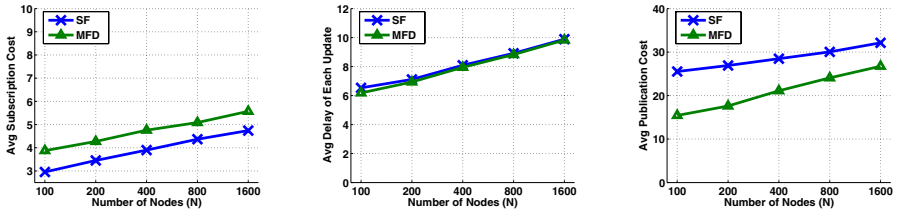
c) Avg update delay

Figure 9. Performance of SF and MFD with 4 different combinations

the highest publication cost, we use this combination throughout the experiments. Note that we use log as the x-axis in all figures of this section, hence a line trend implies a logarithmic behavior.

Figure 10a shows the average subscription cost of SF and MFD as the network size increases. The subscription cost of MFD is slightly higher than SF due to the fact that MFD may require extra message exchanges to establish the path from level 1 to higher level subscription at initial stage. Figure 10b shows the average update delay. Both approaches give similar average update delay, with MFD a slightly better performance. This shows that while striving for low publication cost, MFD does not sacrifice the performance of update delay.

Figure 10c shows the average publication cost of both approaches. Compared to SF, MFD significantly reduces the average publication cost by $20 - 40\%$ in medium sized networks. Although the decomposition of filters may produce more local updates than those in SF, the multi-level structure that we constructed is effective in pruning the filter space. However, as the network size increases, and so does the average hop counts between any two nodes, the pruning of the filter space may no longer compensate the effect of possibly more local updates. We expect the improvement of



a) Avg subscription cost b) Avg update delay c) Avg publication cost

Figure 10. Performance of SF and MFD as the network size increases

MFD to become insignificant when the network size increases to millions.

To summarize, the logarithmic behavior in all three performance metrics demonstrates good scalability of both approaches. In a medium-sized data dissemination network where data publication is much more frequent than subscriptions, MFD gives a much better performance. To better understand the real life implication of these performance figures, based on the result from Figure 10c, a network of 1,000,000 nodes with 200 data sources and an event rate of 1200 subscriptions per hour, publication cost could be kept under 60 messages. If we assume a few milliseconds single hop delay on the internet, the publication delay could be well kept under half a second, which demonstrates the feasibility of our system.

6. Conclusion and Future Work

We have proposed a State-Aware Data Dissemination Network (SA-DDN) to address the problem of dynamic filters. Our simulation suggests that the approach is feasible with reasonable expected delay. Although not currently implemented in SA-DDN, load balancing could be achieved under significant traffic churn by dynamically adjusting the length of the ring segments assigned to each data source. It would be interesting to model the expected performance of SA-DDN analytically. A random walk model of data streams could be a good starting point for such a probabilistic analysis.

The separation of logical structure and physical structure in SA-DDN makes it relatively easy to adapt our approach to other research contexts. In particular, in distributed sensor networks, Chord is no longer the reasonable choice for the underlying network structure: Chord assumes uniform hop delays between Chord nodes, and this is usually not the case in sensor networks. Observe that the geographic routing in sensor networks achieves a $O(\sqrt{N})$ performance, which makes the publication cost even higher. In this case, reducing the publication cost for efficient data dissemination is even more crucial. Our multi-level filter decomposition method could help to achieve such a goal.

References

[1] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Proceedings of IEEE ICDCS*, 2005.

[2] P. Cao and C. Liu. Maintaining strong cache consistency in the world wide web. *IEEE Transactions on Computers*, 47(4), 1998.

[3] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8), 2002.

[4] P. Ganesan and G. S. Manku. Optimal routing in chord. In *Proceedings of ACM SODA*, 2004.

[5] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Proceedings of ACM Middleware*, 2004.

[6] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of ACM SIGMOD*, 2003.

[7] P. R. Pietzuch and J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *Proceedings of DEBS*, 2003.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[9] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of ACM Middleware*, 2001.

[10] S. Shah, K. Ramamritham, and P. Shenoy. Resilient and coherence preserving dissemination of dynamic data using cooperating peers. *IEEE Transactions on Knowledge and Data Engineering*, 16(7), 2004.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

[12] P. Triantafillow and I. Aekaterinidis. Content-based publish/subscribe over structured p2p networks. In *Proceedings of DEBS*, 2004.

[13] H. Wu, B. Salzberg, and D. Zhang. Online event-driven subsequence matching over financial data streams. In *Proceedings of ACM SIGMOD*, 2004.

[14] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering server-driven consistency for large scale dynamic web services. In *World Wide Web*, 2001.

[15] H. Yu and A. Vahdat. Efficient numerical error bounding for replicated network services. In *The VLDB Journal*, 2000.

[16] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment, 2003.

[17] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of NOSSDAV*, June 2001.