Adaptive Reorganization of Coherency-Preserving Dissemination Tree for Streaming Data

Yongluan Zhou Beng Chin Ooi Kian-Lee Tan Feng Yu
National University of Singapore

Abstract

In many recent applications, data are continuously being disseminated from a source to a set of servers. These servers are typically organized into one or more dissemination trees *to enhance the dissemination efficiency. In this paper, we propose a cost-based approach to construct dissemination trees to minimize the average* loss of fidelity *of the system. Our cost model takes into account both the processing cost and the communication cost. To adapt to inaccurate statistics, runtime fluctuations of data characteristics, server workloads, and network conditions etc., we propose a runtime adaptive scheme to incrementally transform a dissemination tree to a more cost-effective one. This scheme employs distributed decisions made by servers independently and is based on localized statistics collected by each server at runtime. The cost model is also extended to incorporate the adaptation overhead. Given apriori statistics of the system, we propose two algorithms to construct a dissemination tree for relatively static environments. These static trees can also be used as initial trees in a dynamic environment. The performance study shows that the adaptive mechanisms are effective in a dynamic context and the proposed static tree construction algorithms perform close to optimal in a static environment.*

1. Introduction

In many emerging monitoring applications (e.g. stock tickers, financial monitoring, network management, sensor networks, traffic control etc.), data occurs naturally in the form of active continuous data streams. These applications stimulate an interest in developing techniques to disseminate data at a very fast rate to a large number of users. Examples of these applications include fast changing data object dissemination [17, 16], streaming XML messages dissemination [10], and publish/subscribe systems [11]. Thus, it is critical and challenging to design an effective, efficient and scalable dissemination system.

In this paper, we look at the design of an adaptive distributed dissemination system, where a data source continuously disseminates fast changing data objects (e.g., sensor data, stock prices and sport scores) to clients via a set of servers. In such a system, clients would submit queries (e.g., continuous queries monitoring the streams or ad hoc queries on the historical and current status) to the servers with their own preferences on data coherency requirements. Based on the requirements of the running queries, each server would have its own interesting object set as well as its coherency requirement of each interesting data object. Like [17, 16], the servers are organized into one or more dissemination trees (with the data source being the root node) so that data/messages are transmitted to each server through its ancestors in the dissemination tree. Each node of the tree would selectively disseminate only interesting data to its child nodes by filtering out the unnecessary ones.

The dissemination efficiency is evaluated using the metric *fidelity* [17, 16]. It measures the portion of time that the values in the servers conform to their coherency requirements. The loss of fidelity at each server is due to the dissemination delay of the update messages, which includes the communication delay as well as the processing delay in its ancestor servers. Interestingly, while it is important to design optimal dissemination trees in this context, there is very little study on this subject.

In this paper, we present a cost-based approach to adapt dissemination trees in the midst of a dynamic changing environment. Our contributions include:

- We formalize the problem by formally defining the metric (fidelity) used to measure the effectiveness of the system and the objective of the whole system (i.e., minimize the average loss of fidelity over all the servers).
- We propose a novel and thorough cost model which considers both the processing cost in the dissemination servers as well as the communication cost in the network links. With the cost model, we can explore a larger solution space than existing methods do to achieve a more cost-effective scheme.
- Based on the cost model, we propose an adaptive runtime scheme that is robust to inaccurate statistics and runtime changes in the data characteristics (e.g., data arrival rates) and system parameters (e.g., workloads, bandwidths etc.). The proposed scheme enables nodes to independently make decisions based on localized statistics collected from neighbouring nodes to transform a dissemination tree from one form to a more cost-effective one. Furthermore, we extend the cost model to incorporate the adaptation overhead. Given apriori statistics of the system characteristics, we propose two static optimization algorithms to build a dissemination tree for rel-

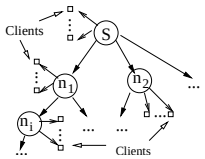


Figure 1. The system architecture

atively static systems. These static trees can also be used as initial trees in a dynamic environment.

• We conducted an extensive performance study which shows that the proposed tree construction scheme performs close to optimal, and the adaptive scheme is also robust to changing conditions at runtime.

The rest of this paper is organized as follows. Section 2 presents the problem and motivations. In Sections 3 and 3.4, we present our solution to the single object dissemination problem, and its extension to the multi-object dissemination problem respectively. A performance study is presented in Section 4. We review related work in Section 5. Finally, we conclude in Section 6.

2. Problem Formulation and Motivations

Figure 1 shows the overview of the architecture of our system. In the system, there is a data source s that stores a set of data objects $O = \{o_1, o_2, \cdots, o_{|O|}\}$, a set of servers $N = \{n_1, n_2, \cdots, n_{|N|}\}$, and a large number of clients. Each server n_i is a continuous stream processing system, such as TelegraphCQ, Aurora and STREAM etc. Each client submits queries involving a subset of data objects through a server (or the data source), and specifies a preference on the coherency on each data object. In this paper, a user's coherency requirement (cr) on a data object is specified as the maximum tolerable deviation from its exact value. From the system's point of view, each server n_i can be viewed as a super-client that requests a subset of data objects O_i from the source, which should be the union of the objects that are requested by the queries running on n_i, and the coherency requirement $cr_{i,x}$ of n_i on object o_x is equal to the most stringent requirement of its queries that involve o_x.

To ensure scalability, we model a generic dissemination scheme as follows. The servers N together with the source s compose an overlay network which can be modeled as a directed complete graph $G = (V, E)$, where $V = N \cup \{s\}$ and E consists of the directed arcs connecting each pair of nodes in V. To build an efficient dissemination scheme, the nodes in V are organized into one or more overlay dissemination tree T. Each T is composed by s, a set of nodes $V' \in N$ and arcs $E' \in E$. The root of all the trees is the source s. Once new values of the data objects at s arrive, s would initiate the messages and disseminate only the necessary ones to each of its child servers in all the dissemination trees. Upon receiving a message, a server would also selectively disseminate it to its child servers. This process happens in each server until the messages reach the leaf servers. Since it is possible for a server's coherency requirement to be less stringent than that of its descendants, every server n_i has an *effective* coherency requirement $cr^m_{i,x}$ on an object o_x which corresponds to the most stringent one among all the $cr_{i,x}$s of the subtree rooted at n_i. A parent performs the filtering of messages based on the $cr^m_{i,x}$ values of its children. In addition to disseminating messages to the child servers, a server that receives a message also has to check whether any of its clients' coherency requirements are violated. If so it would update the results of the query submitted by those clients. In this paper, we assume that clients are pre-allocated to certain servers, and focus on the construction of dissemination trees composed only by the servers and the source. Henceforth we would use "server" and "node" interchangeably and would only consider the dissemination within the dissemination trees.

Following [17, 16], we adopt the notion of *fidelity* as a measure of the performance of a dissemination system. Informally, the fidelity on a data object at a node during an observation period is defined as the percentage of time that the data value at that node conforms to the coherency requirement. To build our cost model, we formulate this metric in a formal way as follows. Let the value of a data object o_x at time t at the source and a node n_i be $o_x(s,t)$ and $o_x(n_i,t)$ respectively, and the coherency requirement of n_i on o_x be $cr_{i,x}$. Then the fidelity of n_i on data object o_x at time t is defined as:

$$f(n_i, o_x, t) = \begin{cases} 1 & : \quad |o_x(s,t) - o_x(n_i,t)| < cr_{i,x} \\ 0 & : \quad |o_x(s,t) - o_x(n_i,t)| \geq cr_{i,x} \end{cases} \quad (1)$$

And the fidelity of n_i on o_x during the observation period $[t1, t2]$ can be computed as

$$F(n_i, o_x, t1, t2) = \frac{\int_{t1}^{t2} f(n_i, o_x, t)}{t2 - t1}.$$

If our observation period is the whole life of the system, it can be rewritten as $F(n_i, o_x)$. Furthermore, the average fidelity at node n_i is computed as

$$F(n_i) = \frac{1}{|O_i|} \sum_{\forall o_x \in O_i} F(n_i, o_x).$$

The *loss of fidelity (LF)* is defined as the complement of fidelity, which is $LF(n_i) = 1 - F(n_i)$. Our objective is to minimize the average loss of fidelity over all nodes

$$AvgLF = \frac{1}{|N|} \sum_{i=1}^{|N|} LF(n_i).$$

Since the loss of fidelity is due to the delay of the messages, we adopt an eager approach: the source node continuously pushes update messages to child servers as soon as the corresponding coherency requirements are violated, and each server, upon receiving any update messages, also pushes the necessary ones to its children as soon as violations occur.

We define the *Min-AvgLF problem* formally as follows: *Given a source s, a set of data objects O, a set of servers N, and the set of requesting data objects O_i of each server n_i as well as the coherency requirement $cr_{i,x}$ of n_i on each $o_x \in O_i$, construct/adapt one or more dissemination trees T to minimize the average loss of fidelity ($AvgLF$) of the system.*

By the celebrated Cayley's theorem, the number of spanning tree of a complete graph is $|V|^{|V|-1}$, where $|V|$ is the number of nodes in the graph. This means that brute-force searching is prohibitive even for a moderate number of nodes (e.g. 16 nodes). Even worse, a simpler problem is already shown to be NP-Hard [5].

In view of the complexity of the problem, existing approaches such as DiTA [16] adopt two heuristics: (a) the coherency requirement of a parent node is at least as stringent as its children; (b) Each server has an apriori constraint on the fanout, i.e., the maximum number of child servers is predetermined. However, under these restrictions, the resulting dissemination tree would be far from optimal. This is because they ignore the differences of the servers in their capabilities as well as their communication delays. For example, although a server has a slow CPU, a low bandwidth or a high workload, it would still be put at the upper level of the tree as long as its coherency requirement is relatively stringent. However, all its descendants suffer the long processing delay in the slow server. This would result in severe loss of fidelity. To handle these limitations and find out the tradeoffs, we believe a cost-based approach that captures both communication and processing cost is likely to lead to a more cost-effective dissemination tree.

Yet another challenge is that the optimality of a dissemination scheme is dependent on the current system parameters (such as data arrival rates, system workloads etc.). However, in a large scale distributed system, this information is hard to estimate or collect beforehand. Moreover, these parameters would fluctuate over time. For example, users would change their coherency requirements; a server's workload would change as the number of clients connected to it are increased or decreased, or the message rate of each server would also change due to the fluctuation of the data values. Since the dissemination system runs continuously, it can experience these changes at runtime, which would make the previously optimal scheme suboptimal. The problem of adapting to inaccurate statistics and system changes has been extensively explored in other problems such as query processing [15][2]. Unfortunately, few efforts have been devoted to adapting the structure of a dissemination tree at runtime. Moreover, a decentralized scheme is highly preferable due to scalability and reliability problems.

3. Coherency-Preserving Dissemination Tree

In this section, we look at the scheme to construct a tree T to disseminate data objects. We note that T is a spanning tree of the overlay graph G. We first focus on single object dissemination and present the cost model to evaluate the LF of a tree T, then describe the runtime adaptation scheme and present the two static tree construction schemes. All the algorithms proposed do not place any restriction on the maximum fanout allowed; neither does it require the internal nodes to be more stringent in the coherency requirements than its child nodes. Finally the techniques are extended to multi-object dissemination.

3.1. Cost Model

In a cost-based approach, a cost function is used to evaluate the goodness of a potential solution. In our case, we propose a novel cost model to measure the LF of a dissemination tree. In the cost model, we make the following assumptions and simplifications:

1. A message sent from n_i to n_j incurs a communication delay, whose expected value is denoted as $d(n_i, n_j)$.

2. The messages received by a node are processed in a FIFO manner. Upon receiving a message, n_i would check every child to see whether the message should be disseminated to it. The processing order of the children is assumed to be random. Let the time to perform the filtering be t_i^p and the time to perform the transmission be t_i^c. t_i^c includes the time to package the message and the time to send out the packages. The latter part is inversely proportional to the available bandwidth of n_i.

3. Each node would assign a portion of its resources (e.g. CPU, bandwidth, etc.) to perform the task of disseminating data to its child nodes. This portion of resources might be adjusted periodically. However, within each period, we assume it is fixed. Furthermore, the workload of a node is defined as the fraction of time that the node is busy.

Given these assumptions, now let us see how to estimate the loss of fidelity of a node n_i. The LF of n_i arises because of the delay of an update message. If the number of messages per unit time (i.e., the average message arrival rate) for n_i is r_i and the average delay of each update message is D_i, then the average LF of n_i is $LF(n_i) = r_i \cdot D_i$. r_i is related to the data characteristics and the coherency requirement of n_i. Now we need to estimate D_i. At a closer look, D_i includes the communication delay in all the links and the processing delay in all the nodes along the path from the root to n_i. To compute the communication delay, we define $D(n_j, n_i)$ as the communication delay from n_j to n_i in the dissemination tree T. It is obvious that $D(n_j, n_i)$ is the sum of the communication delay of the overlay edges in the unique path from n_j to n_i. Hence the total communication delay of a message from s to n_i is $D(s, n_i)$. In the following paragraphs, we present how to estimate the second part of the delay: the processing delay.

The processing delay of a message for n_i in each of its ancestor n_k can be divided into the queueing time and the processing time. Let us estimate them one by one.

1. Queueing time. In our model, each node is a queueing system. From basic queueing theory, the expected queueing time of a message in a M/M/1 system is equal to $\frac{\rho}{1-\rho}t$ where ρ is the workload of the system and t is the expected processing time of a message. The workload of the system is equal to the message arrival rate times the expected per-message process time t. Hence to estimate the queueing time, we have to estimate the expected per-message processing time. Note that our tree construction scheme does not require the coherency requirement of a parent node to be more stringent than that of its descendant nodes. Thus, every node has an effective coherency requirement cr_i^m, which should be the most stringent crs within the subtree rooted at n_i. Consequently, there

is an effective message arrival rate r_i^m for n_i, which should be equal to the maximum message arrival rate within the subtree rooted at n_i. For each message arrived at a node n_k, the probability that it is sent to a child n_j is r_j^m/r_k^m. Hence the expected processing time of a message in n_k for each of its children n_j is

$$t_{kj} = t_k^p + t_k^c \frac{r_j^m}{r_k^m}. \quad (2)$$

Therefore, if we denote the set of child nodes of n_k as C_k, then the expected processing time of a message in n_k can be estimated as:

$$t_k = \sum_{n_j \in C_k} t_{kj}. \quad (3)$$

Given t_k, the average processing time of a message, we can derive that the workload of n_k is $\rho_k = r_k^m t_k$. Hence the queueing time of a message in node n_k is $\frac{\rho_k}{1-\rho_k} t_k$. Note that this covers both the queuing times for processing and transferring a message.

2. Processing time in n_k for a message received by n_j. Since the children are processed in random order, before checking a child node n_j, there are on average $(|C_k| - 1)/2$ other children that have been processed. The expected length of this time is equal to $(1/2)(t_k - t_{kj})$. Then it takes a t_k^p time to check for n_j and then takes another t_k^c time to transmit the message to n_j. This means that the expected processing time in n_k for a message received by n_j is $(1/2)(t_k - t_{kj}) + t_k^p + t_k^c$.

Summing up the queueing time and the processing time, we can derive the processing delay in n_k for a message received by n_j as

$$g(n_k, n_j) = \frac{1+\rho_k}{2(1-\rho_k)} t_k + t_k^p + t_k^c - \frac{1}{2} t_{kj}. \quad (4)$$

This function can accurately estimate the processing delay. However, it distinguishes the delays for different children, which will bring higher cost in our algorithm. Hence we propose an approximation, where we use the average processing delay over all the children, to approximate the delay for each of them. Simple calculations can derive that this processing delay is

$$\begin{aligned} g(n_k) &= \frac{1}{|C_k|} \sum_{n_j \in C_k} g(n_k, n_j) \\ &= \frac{1+\rho_k}{2(1-\rho_k)} t_k + t_k^p + t_k^c - \frac{1}{2|C_k|} t_k \end{aligned} \quad (5)$$

Now, we would derive the cost function to estimate the loss of fidelity for a node n_i as

$$\begin{aligned} LF(n_i) &= r_i \times [D(s, n_i) + g(p(n_i)) + g(p(p(n_i))) \\ &\quad + \cdots + g(s)] \end{aligned} \quad (6)$$

where $p(n_i)$ denotes the parent of n_i.

3.2. Adaptive Reorganization of Dissemination Tree

In this subsection, we present our runtime scheme that adaptively reorganizes a given dissemination tree to a more cost-effective one. The algorithm is a distributed local search scheme. At each state, distributed nodes would search the neighbor states that can improve the current state. Neighbouring states are generated based on a set of transformation rules extended from [4]. In the following subsections, we first present the local transformation rules that specify how the states could be transformed and how to estimate the benefit of the transformations. Then we present how to efficiently make adaptation decisions. Finally we summarize the set of information that has to be collected at runtime to support the adaptive scheme and present how to extend the cost model to incorporate the adaptation cost.

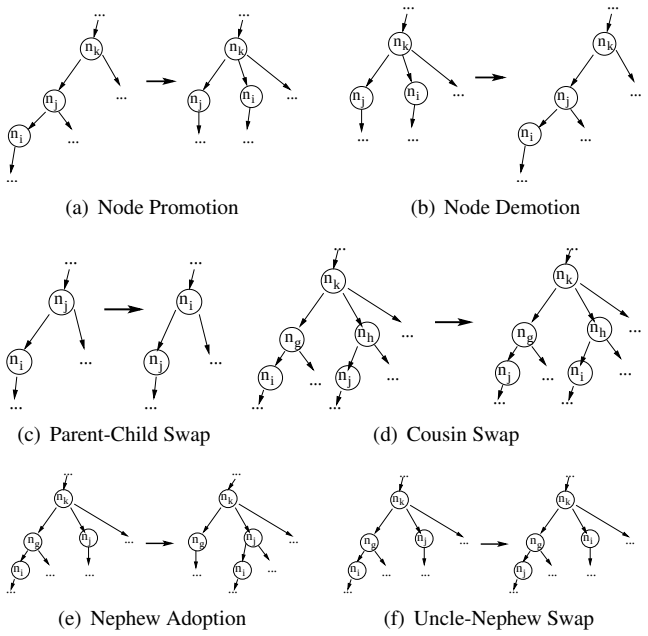


Figure 2. Local Transformation Rules

3.2.1. Local Transformation Rules. In this section we define several local transformation rules that transform a scheme into its neighbor schemes. We have identified six rules. In all these rules, when a node is moved, all its descendants would be moved along with it.

1. **Node Promotion**: Promote a node n_i to its parent's sibling. Figure 2(a) shows an example of this transformation. In the example, n_i is promoted to a sibling of its previous parent n_j. This transformation might be beneficial, for example, when the workload of n_k is reduced as a result of a decrease in the number of its clients and hence more of its resources are assigned to the dissemination task. Promoting n_i can reduce the communication delay of messages sent to n_i and all its descendants if $d(n_k, n_j) + d(n_j, n_i) > d(n_k, n_i)$. This would also be helpful if we underestimate the capacity of n_k when building the initial dissemination tree.

2. **Node Demotion**: Demote a node n_i to a child of one of its siblings. In the example shown in Figure 2(b), n_i is demoted to the child of its prior sibling n_j. This transformation may be beneficial, for example, when n_k's workload is increased and hence less resources are assigned to the dissemination task. Demoting n_i can reduce the dissemination load of n_k and hence reduce the processing delay of messages to be sent to the descendants of n_k. In addition, it also helps to

handle any overestimation of the capacity of n_k in the initial tree building.

3. **Parent-Child Swap**: Swap the positions of n_i and its parent. In Figure 2(c), the positions of n_i and its parent n_j are swapped.

4. **Cousin Swap**: Swap the position of two nodes n_i and n_j which have the same grandparent n_k. Figure 2(d) shows an example.

5. **Nephew Adoption:** A node n_h adopts its *nephew* n_i and adds it as its own child. As shown in Figure 2(e), n_i's grandparent is the parent of n_h. In this transformation, n_i is added as a child of n_h.

6. **Uncle-Nephew Swap:** Swap the positions of n_h with its nephew n_i. Figure 2(f) depicts an example.

Actually the first two basic transformation rules are complete, i.e. any other transformations can be composed based on these two transformations. For example, Nephew Adoption can be composed by first promoting n_i and then demoting it to a child of n_h. However, using composite transformations directly may help avoid being stuck in a local optimum. The four composite transformations that follow are proposed based on this intuition. While the composite transformations can be extended to involve arbitrary nodes, we only consider these transformations because the runtime adaptation scheme can be relatively simple and less costly.

Based on our cost model, we can recompute the cost of the dissemination tree after the transformations, which will take $O(|N|)$ time. But since the transformations only affect part of the tree, rather than computing the cost from scratch, we can compute the change of the cost in *constant time*. Here we would use Node Promotion to illustrate. The computation of the other transformations can be found in the full version [19].

As depicted in Figure 2(a), node n_i is to be promoted, and n_j and n_k are the parent and grandparent of n_i respectively prior to the transformation. After the transformation, the messages to be sent to n_i would no longer experience the transmission delays $d(n_k, n_j)$ and $d(n_j, n_i)$, and the processing delay in n_j. However it would experience the new transmission delay $d(n_k, n_i)$. This would also affect all the nodes below n_i. Hence this results in the change of $AvgLF$ which is

$$\begin{aligned}\Delta AvgLF_1 &= \frac{1}{|N|} r_i^c [d(n_k, n_i) - d(n_k, n_j) \\ &\quad - d(n_j, n_i) - g(n_j)],\end{aligned}$$

where r_i^c is the aggregated message rate over all nodes in the subtree T_i rooted at n_i, i.e. $r_i^c = \sum_{n_p \in T_i} r_p$. Furthermore, the load in n_k and n_j would be changed after the transformation. Hence all the nodes below them would experience the change of the cost due to the load changes. This results in the change of $AvgLF$ which is

$$\begin{aligned}\Delta AvgLF_2 &= \frac{1}{|N|} \left\{ (r_j^c - r_j)[g'(n_j) - g(n_j)] \right. \\ &\quad \left. + (r_k^c - r_k)[g'(n_k) - g(n_k)] \right\},\end{aligned}$$

where $g'(n_j)$ and $g'(n_k)$ denote the estimated new processing delay in n_j and n_k respectively if the transformation is to have taken place. $\Delta AvgLF$ is equal to the sum of $\Delta AvgLF_1$ and $\Delta AvgLF_2$. Other transformations can be analyzed similarly.

3.2.2. Adaptation of Dissemination Tree. The adaptation scheme works as follows: periodically, compute the benefit (i.e., $(-1) \cdot \Delta AvgLF$) of each possible transformation, and then perform those that have positive benefits. To implement this procedure, there are several choices. In one extreme, we can select a server to act as a centralized controller to make the adaptation decisions. However, as discussed, this approach suffers from problems of scalability and reliability. In another extreme, we can design a totally distributed approach. In this approach, each node makes the decisions independently and asynchronously. However, this totally unstructured scheme would result in (a) Conflicting decisions being made by different nodes, e.g., n_i may determine to promote itself and meanwhile its parent may want to swap with it. Extra mechanisms have to be employed to resolve this problem, potentially increasing the complexity of such a scheme. (b) Wastage of computational resources as a result of multiple nodes arriving at the same decisions, e.g., n_i and its parent determine to swap with each other at the same time.

To alleviate these problems, we propose a more structured mechanism. The adaptation operates in rounds. The root node initiates each round by creating a token. Only when a node holds a token, could it make an adaptation attempt. Algorithm 1 presents the operations to be executed in a node that receives a token. Each node receives a token can make its own decision independently without any synchronization with the other nodes. Instead of allowing every node attempts to try all kinds of transformations, we restrict that each node would only consider the transformations involving its children and grandchildren. These include promoting a grandchild (node promotion), demoting a child (node demotion), swapping a child and a grandchild (parent-child swap and uncle-nephew swap), swapping two grandchildren (cousin swapping), and moving a grandchild from one child to another child (nephew adoption). A node sends reorganization requests (if any) to the involved descendants: i.e. n_j in both Fig. 2(a) and (b), n_i and n_j in Fig. 2(c), n_g and n_h in both Fig. 2(d) and (e), n_g in Fig. 2(f). After the adaptation (if any) has been carried out, a copy of the token is sent to each of its non-leaf children. The next round of adaptation would initiated by the root node if the adaptation interval is exceeded. If a node receives a token when it is still doing an adaptation, it would just ignore the token. Furthermore, if a node receives a reorganization request if it is holding a token then it would also ignore the reorganization request to avoid any contradictions.

3.2.3. Information Collection. Given the adaptation scheme described above, we now look at what information should be collected at runtime. Since each node would only consider transformations involving its children and grandchildren, it would collect state information from its children and grandchildren. Hence a node contains at most the information of $O(C^2)$ nodes, where C is the maximum out-degree of all nodes. The information to be collected has to enable us to

Algorithm 1: AdaptationAttempt

```
begin
    maxBenefit ← 0; t ← NULL;
    for each possible transformations t1 involving the
    children and grandchildren do
        if maxBenefit < Benefit(t1) then
            maxBenefit ← Benefit(t1);
            t ← t1;
    if t ≠ NULL then Perform t;
    for each child n_j do
        if n_j is not a leave node then
            Send one copy of the token to n_j;
end
```

calculate the benefit of the transformations. Specifically, the information stored in a node n_i is as follows:

1. The overlay paths from n_i to its children and grand-children. This information once obtained is not necessary to be collected at runtime. Because any change in the structure in this part is determined by n_i itself and n_i updates the information itself.
2. The values of r_j^m, r_j^c, as well as t_j^c and t_j^p of each of its children and its grand-children.
3. The value of r_i^c. Actually, r_i^c can be computed based on the r_j^c value stored in each child node n_j, i.e. $r_i^c = (\sum_{n_j \in C_i} r_j^c) + r_i$.
4. The physical communication delay between n_i and each of its children or grand-children, and those between each of its children and each of its grand-children.

The information collection scheme is also a window-based scheme. Each node asynchronously maintains its own information collection window. At the end of each window, a node would measure the necessary information. If it detects that the new value is increased to (1+τ) times or decreased to 1/(1+τ) times of its previous value, it would send the new value to its parent. In our experiments, we set τ to be 0.2.

3.3. Static Tree Construction Algorithms

In this subsection, we present two static tree construction algorithms: a greedy algorithm and a randomized algorithm based on Simulated Annealing[14]. Given apriori statistics on the system parameters, the two algorithms can generate a good dissemination tree. Such a tree can be used in environments that are static and not subject to runtime changes. For a highly dynamic environment, the algorithms provide a good initial scheme (as compared to a randomly generated dissemination tree) that can speed up the convergence to the optimal scheme as dissemination trees are refined adaptively based on the runtime characteristics.

3.3.1. Greedy Algorithm. The algorithm is presented in Algorithm 2. It adopts a greedy heuristic. The algorithm sorts the nodes in ascending order of $d(s, n_i)+t_i^p+t_i^c$. Then it adds the nodes into the dissemination tree one by one in the sorted order. The partially built dissemination tree T is represented as the set of nodes and edges in the tree. For each new node $N[i]$, it selects one node n_j within the partially built tree to act as the parent of $N[i]$ so that the average loss of fidelity $AvgLF$ of the new tree $T \cup \{N[i], e(n_j, n_i)\}$ is minimized. The estimation of $AvgLF$ is based on Equations (3), (5) and (6). To save the computational time, simple techniques can be employed to compute the new $AvgLF$ value incrementally based on the current $AvgLF$ of the partial tree. Due to space limit, we do not present the details here. Given each potential parent, it takes $\log |N|$ time to estimate the new $AvgLF$. Therefore, the computational complexity of Algorithm 2 is $O(|N|^2 \log |N|)$.

Algorithm 2: Greedy

```
begin
    Add s to T;
    N[0] ← s;
    N[1 ··· |N| − 1] ← Sort the other nodes in ascending
    order of value d(s, n_i) + t_i^p + t_i^c;
    for i = 1; i < |N|; i++ do
        e ← arg min_{0≤j<i} AvgLF(T ∪ {N[i], e(n_j, n_i)});
        Add N[i] and e to T;
    return T;
end
```

The dissemination tree built by using this algorithm has the following property:

Theorem 1 *If the height of the tree is h, and the delay between pairs of nodes satisfy the triangle inequality*[1]*, then the communication delay of a message received by n_i is at most $2d_i \cdot h$ where $d_i = d(s, n_i) + t_i^p + t_i^c$. Further assume that the fanout of each node is at most C and the maximum message rate over all nodes is at most r, then the processing delay of a message received by n_i is at most*

$$h \cdot \left(\frac{1 + r \cdot C \cdot d_i}{2(1 - r \cdot C \cdot d_i)} C \cdot d_i + d_i\right)$$

The proof can be found in the full version of this paper[19].

3.3.2. Simulated Annealing. Since the Min-AvgLF problem is NP-Hard, we use a probabilistic approach, Simulated Annealing[14](SA), to approximate an optimal solution. This approach has been shown to generate very efficient solutions for hard problems, such as large join query optimizations [12]. The algorithm is illustrated in Algorithm 3. It starts from a random scheme S_0 and an initial temperature T_0. In the inner loop, a new scheme $newS$ is chosen randomly from the neighbors of the current scheme S. If the cost of $newS$ is smaller than that of S, the transition will happen. Otherwise, the transition will take place with probability of $e^{-\Delta C/T}$. (With the decrease of T this probability would be reduced.) Meanwhile, it also records the minimum-cost scheme that has been visited. Whenever it exits the inner loop, the current temperature would be reduced. Based on our experimental tuning and past experiences[13][12],

[1] If every non-leaf node has at least 2 children, then $h \leq \log |N|$. In addition, some studies [18] have shown that violations of triangle inequality is not very frequent, which is only about $1.4\% \sim 6.7\%$.

we select the parameters as follows: (1) T_0: $2 * cost(S_0)$; (2) $frozen$: $T < 0.001$ and $minS$ unchanged for 10 iterations; (3) $equilibrium$: $64 \times \#nodes$; (4) $reduceTemp$: $T \leftarrow 0.95T$; (5) $RandomNeighbor$: randomly choose one of the transformations listed in Section 3.2. The cost of the new scheme can be computed using the incremental cost computation presented in Section 3.2. Given a static environment and accurate system parameters, we believe this algorithm can derive the best dissemination scheme over all the other algorithms. However, its optimization overhead may be high. Moreover, such a centralized scheme will incur too large a communication overhead in a dynamic context.

Algorithm 3: Simulated Annealing

```
begin
    S ← S_0; T ← T_0; minS ← S;
    while !frozen do
        while !equilibrium do
            newS ← RandomNeighbor(S);
            ΔC ← cost(newS) − cost(S);
            if ΔC ≤ 0 then S ← newS;
            else S ← newS with probability e^{−ΔC/T};
            if cost(S) < cost(minS) then minS ← S;
        T ← reduceTemp(T);
    return minS;
end
```

3.4. Multi-Object Dissemination

In the above discussion, we only consider single object dissemination. To disseminate multiple objects, there are two possible solutions: (a) the multiple tree approach (to build one dissemination tree for each data object), and (b) the single tree approach (to build one tree for multiple data objects). However the multiple tree approach would consume more resources and incur higher maintenance cost. Hence we would focus on the single tree approach in this paper and postpone the multiple tree approach as our future work. In the single tree approach, a single dissemination tree T is built to disseminate a set of objects. Note that if an object of interest to a child is not requested by the parent itself, the parent's requesting object set would be enlarged to include this object. Hence there is an effective object set O_i^m for a node n_i which is the union of all the interesting objects of the nodes in the subtree rooted at n_i.

Due to the space constraint, here we only briefly present how the above techniques can be extended to the multi-object case. We refer the readers to the full version [19] for more details. First, we need to extend the the cost model. The derivation process of the cost model is similar to the single object case, except that we have to deal with more than one object. As in the single object case, we also need to design an adaptive scheme and a static scheme. For the adaptive scheme, the transformation rules as well as the adaptation mechanism are also the same as the single object case. However, we need to extend the information collection strategy to include the new information that are required by the new cost model. Both the Greedy and SA Algorithm can be used here by employing the new cost model. The complexity of Algorithm 2 becomes $O(|O| \cdot |N|^2 \cdot \log |N|)$. Theorem 1 can also be applied to this scheme. Note that, in this case, the parameter r in the theorem would be the sum of the maximum message rate among all the nodes for each data objects.

4. Experiments

In this section, we present a performance study of the proposed techniques, and report our findings. The simulator is implemented using ns-2, a popular discrete-event simulator for networking research. The topology is generated using the GT-ITM topology generator. The Transit-Stub model, which resembles the Internet structure, is used. We generate a network topology with 1500 nodes, of which one node is chosen as the source, 256 nodes are selected as the dissemination servers, and the remaining nodes act as routers. The average communication delay between any two servers is about 20ms. The expected filtering time and transmission time of each node is derived by using two respective uniform distribution. In our basic configuration, we set the average values of these times as 5ms and 1ms respectively (which may vary in our experiments), and set the minimum values as 1ms and 0.125ms respectively. The source server's expected filtering time and transmission time are always set to the minimum value to model an enterprise class server. Given the expected filtering time t_i^p and transmission time t_i^c for a node, the exact filtering time and transmission time of each message are drawn from two respective exponential random variable with expected values as t_i^p and t_i^c respectively. Recall that each server in our system has to process local user queries (probably complex queries) and disseminate data to the child servers, and only a limited resource can be allocated for the dissemination task. Hence we use a relatively long filtering time and transmission time which captures the load of processing user queries in the servers. In addition, the adaptation interval of our adaptive scheme is set to 200 seconds and the information update window is set to 50 seconds. We model the time used to transmit the statistical information to be the same as t_i^c. All the experiments are conducted in a Linux server with an Intel 2.8GHz CPU. We also implemented the optimization algorithms and the adaptation functions in C to study their performance. To examine the cost of making adaptation decisions, we use a node that serves 100 objects and try estimating 100 possible decisions. We found that $t_k^d \approx 0.6\mu s$. The cost of collecting information is analyzed similarly. In the following experiments, we set both cost as $1\mu s$ in the simulation.

To evaluate the performance of the proposed techniques, we compare them with the following approaches:

1. **DiTA**[16]. In DiTA, a tree is constructed for each data object. Fanout constraint is set for each server to avoid overloading. In our experiments, this is done by trial-and-error by repeatedly trying with different parameters and to pick the set that gives the optimal performance.(We find that this is the only way to find good fanout constraints and we believe

this is a disadvantage of schemes relying on predetermined fanout constraints.) Furthermore, we use a centralized version of DiTA biased towards DiTA. It first sorts the nodes in ascending order of the values of their coherency requirements and then adds them one by one into the tree in the sorted order. When adding n_i, a node within the partial tree, which has the smallest communication delay to n_i and still has available fanout degree, is selected as the parent of n_i.

2. **Source-Based Approach**. The distributed servers do not cooperate and all the servers are connected to the source. This provides a base line to evaluate all the schemes.

3. **Random Tree**. The nodes are added in random order. For each joining node, randomly select a node to act as its parent. This scheme provides a base line to evaluate all the tree-based schemes.

Furthermore, in the experiments, we use two types of datasets: synthetic data and real data. In the synthetic dataset, we set a specific expected message rate $r_{i,x}$ for each node on every object based on a uniform distribution. The source is of the largest $r_{i,x}$ for all the objects. Given the $r_{s,x}$ of the source, the interval of each update message is an exponential distributed variable with an average value of $1/r_{s,x}$. The synthetic data set provides relatively steady message rates, which offers opportunities for us to study the properties of the different algorithms. For the real dataset, we continuously poll stock traces from `http://finance.yahoo.com`. The polling is done in an interval of one second. In the experiments, we use 100-500 traces as our basic dataset.

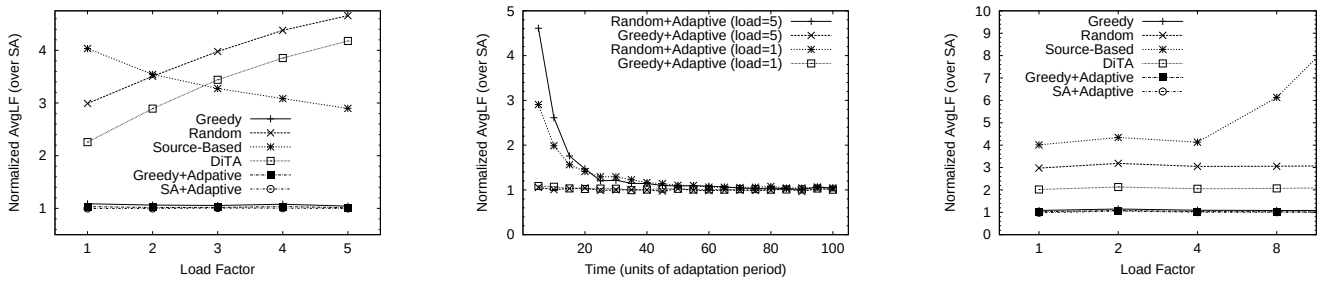
4.1. Static Environment

In this section, we will study how the various algorithms perform in a static environment.

In the first set of experiments, we examine the algorithms in a single object dissemination situation. We utilize the synthetic dataset. The expected message rate of each node is selected from a uniform distribution with the average value of 1 messages/second and a minimum value of 0.5 messages/second. (Note that the message rate models the coherency requirement at each node - a small coherency requirement implies a high message rate, and vice versa.) We vary the average filtering time and transmission time by multiplying them with a parameter $load$. The parameter $load$ ranges from 1 to 5 in our simulation. The minimum values of filtering time and transmission time are not changed. This models two effects: (1) Various load conditions of the whole system. When more clients are connected or more queries are submitted to a node, its load would become higher and hence it takes a longer time to disseminate messages to its child nodes. The filtering and transmission times of these nodes would be increased. (2) Various degrees of heterogeneity of the system. With a higher value of $load$, the filtering time and transmission time of the nodes would differ to a higher degree. No matter which is the case, nodes with higher filtering and transmission time would be deemed as less capable nodes and hence a good plan should be able to identify this kind of nodes and put them at a lower level of the dissemination tree. We run each algorithm for $20,000$ seconds and record the average $AvgLF$ over the whole simulation period as well as the values within every $1,000$ seconds time window. To ease the comparison, we normalize the $AvgLF$ values of all the other algorithms over that of the SA algorithm, which we believe to be the best dissemination scheme.

Figure 3 shows the results of our experiment. From Figure 3(a), we can see that when $load = 1$, Greedy and the adaptive counter-part (Greedy + Adaptive) perform as well as SA, while the adaptive algorithm slightly improves over the initial scheme. Due to the optimality of SA, the adaptive scheme has few opportunities to further optimize the scheme. On the other hand, DiTA has more than two times $AvgLF$ than SA. That is because it can neither differentiate the capabilities of the different nodes nor utilize information of the communication delays between the nodes. The source-based algorithm performs the worst. In this scheme, all nodes are connected to the source node. Although the source node in our settings is not overloaded, the messages would still experience very long delay in the source node because of the high workload of the source. The random tree algorithm on the contrary scatters the workload randomly over all the nodes, and hence has a smaller $AvgLF$ value. However, with the increase of the $load$ parameter, we can see from Figure 3(a) that the relative performance of the source-based scheme improves. This is because, in our study, increasing the $load$ parameter increases the processing time of all the nodes except the source node. Since the source-based approach disseminates the messages directly from the source, it is not influenced by the $load$ parameter. On the contrary, all the tree-based schemes would suffer from the increase of $load$. Furthermore, with the increase of $load$, DiTA and the random tree scheme become much worse while our static algorithms with/without adaptation scheme remains effective. This is because our scheme can identify the different capabilities of the nodes and reorganize them in a more cost-effective way.

Although our static schemes work well as shown above, they rely on accurate system statistics. To examine the performance of our adaptive mechanisms without these statistics, we use the random scheme to model an initial scheme that would be generated without accurate statistics. Figure 3(b) shows the result of this experiment. To ease viewing, we only depict the results of $load = 1$ and $load = 5$ for the Random+Adaptive and Greedy+Adaptive algorithms. The curves of the other $load$ values would be between these two cases. It can be seen that when there are accurate system statistics, Greedy would result in a good dissemination scheme that works as well as SA. Hence there are not many opportunities for the adaptation scheme to improve. On the contrary, the random scheme works far worse than SA. Our adaptation algorithm iteratively improves this initial scheme. After about 30 adaptation periods, the random scheme has been improved from more than 3 and 4 to only 1.3 times of the performance of SA. And after more adaptation periods, the random scheme is improved to the extent that it performs as well as SA. This clearly shows the need for adaptive strategy, as well as the



(a) Sensitivity to processing time (b) Online improvement (c) Sensitivity to message rate

Figure 3. Performance on single object dissemination

effectiveness of our adaptive scheme.

Another type of load change of the system is the change of message rates. With the increase of message rates, the dissemination load of the system is increased. In this experiment, we fix the processing time of each node to its basic value and multiply each node's basic message rate with the $load$ parameter. The results are depicted in Figure 3(c). With increasing message rate, Source-Based deteriorates rapidly. This is because with a high message rate, the workload of the source node largely increases due to its large number of children, and this incurs long queueing time for the messages in the source node. On the other hand, the relative performances of all the tree-based algorithms are not sensitive to message rate changes. This is due to the moderate number of child nodes in a tree-based scheme. Furthermore, our schemes steadily outperform the others under various message rates.

We also rerun the first two experiments on the multi-object case using our collected stock traces. The results suggest similar conclusions [19]. Furthermore, another experiment is done to examine the sensitivity of the algorithms to different number of data objects. We vary the number of data objects to be disseminated from 100 to 500. The results are depicted in Figure 4. With different number of data objects, Greedy, Greedy+Adaptive and SA+Adaptive persistently outperform all the other algorithms. We can also see that the relative performance of the Source-Based algorithm deteriorates with increasing number of data objects. This is because the source's workload largely increases with increasing number of data objects and hence its processing delay increases. Furthermore, the absolute values of the $AvgLF$s of all the other tree-based algorithms only increase by around 15% when the number of objects is increased from 100 to 500. However, for the $AvgLF$ of Source-Based, the increase is around 200%. This shows that the tree-based approaches have better scalability to the number of objects.

4.2. Dynamic Environment

In this subsection, we study our adaptive algorithm under a dynamic environment. In the experiments, we study how the algorithms perform when the workloads of servers are changed. The first experiment studies the single object dissemination schemes using the synthetic dataset. The parameters are set as in the first experiment in the last subsection where $load = 1$. Since Source-Based and Random have been shown to work worse than the others in this situation, we only examine the results of the other algorithms. We run the system for $20,000$ seconds, and at the $10,000$th second, we increase the processing time of 10 nodes that are the first 10 nodes (except the source node) in a breadth-first search of the dissemination tree. These nodes are at the top of the dissemination tree. Their filtering time and transmission time are increased to 10 times of the previous values. This models the situation that the workloads of some nodes at the higher level of the tree increase as more clients are connected or more queries are submitted. The result is depicted in Figure 5(a). In order to examine the optimality of the algorithms before and after the state transitions, we run the SA algorithm under both conditions. We then normalized the $AvgLF$ value of each algorithm under each condition by the corresponding $AvgLF$ of the SA algorithm. We compute the average of the normalized $AvgLF$ values over a 1000 seconds window and then report the 20 resulting values. In figure 5(a), one can see that at the first $10,000$ seconds, SA and SA+Adaptive perform as well as SA, while DiTA is two times worse than them. After the $10,000$th second, the $AvgLF$s of both DiTA and SA drastically increase. That is because the 10 nodes whose processing times are increased become the bottleneck of the whole dissemination tree. Furthermore because they are at the top of the tree, their processing delays dominate the delays of the messages sent to all their descendant nodes. On the other hand, our adaptive mechanism can detect this change and hence reorganize the dissemination tree to adapt to the new situation. Therefore, it only has a short term increase in the $AvgLF$ and then drops back to the original state. That is because the highly loaded nodes have been put to lower levels of the tree and then their high processing times have little effect on the dissemination efficiency.

We also conducted a similar experiment on multiple object dissemination to examine the adaptivity of our scheme. Since DiTA builds one tree for each object and DiTA has been shown above that it is not adaptable to system changes for any one of its dissemination trees, we only compare the SA and SA+Adaptive in this experiment. Again we use the stock traces as our input dataset. The other settings are similar to the above experiment. At the $5,000$th second, we shift the filtering time and transmission time of 10 nodes, which are at the top of the dissemination tree, to 10 times of their original values. The result is reported in Figure 5(b). We

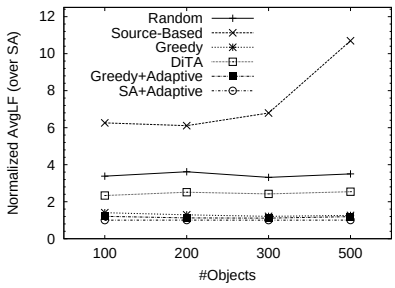
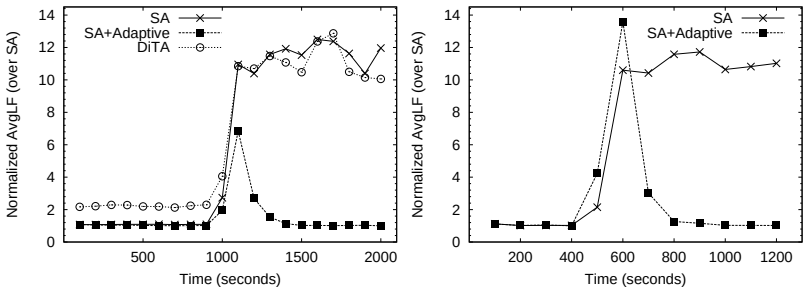


Figure 4. Number of objects



(a) Single object dissemination (b) Multi-object dissemination

Figure 5. Performance in dynamic environment

can see that before the change, SA works slightly worse than Adaptive. At the $5,000$th seconds, both SA and SA+Adaptive increase in their $AvgLF$s. However, our adaptive mechanism successfully detects the shift and then reorganizes the dissemination tree to adapt to the new situation. Hence SA+Adaptive restores back to its original state in terms of $AvgLF$ while the bad performance of SA persists. We also performed experiments on runtime change of transmission delays and coherency requirements. The results show that our adaptive scheme can also adapt to these changes and reoptimize the scheme incrementally.

5. Related Work

In [17, 16], the authors proposed an interesting problem: disseminations streaming data with coherency preserving and two techniques to construct a dissemination tree/graph: LeLA (Level by Level Algorithm)[17] and DiTA (Data item at a Time Algorithm)[16]. DiTA is reported to be much better than LeLA. However, the authors do not provide a cost model. Hence the factors that affect the system performance is unclear and it is hard to measure the optimality of a construction scheme. Furthermore, adaptivity is not addressed.

Application-layer multicast is shown to be much easier to deploy than IP layer multicast with only little penalties in performance [8]. More recently, optimization of application-layer multicast tree is studied in a few pieces of work [4, 6]. However, these systems assume all data would be transferred to every node in the multicast tree and the effect of filterings in the middle of the disseminations is not considered. As can be seen in our cost model, the filtering has very significant effect on the cost of the dissemination tree. Ignoring the filtering effect will result in a scheme far from optimal. Hence they are not adequate for our problem.

Authors in [10] presented the design of a large scale distributed XML dissemination system. Distributed content-based pub/sub systems have also been studied in the networking community [1, 3, 7, 9]. However, most of these efforts focus on how to efficiently filter and route contents to the clients based on the clients' interests. The organization of dissemination servers are not considered.

6. Conclusion

In this paper, we reexamined the problem of designing a scalable dissemination system. We proposed a cost-based approach to construct dissemination trees to minimize the average *loss of fidelity* of the system. Based on our cost model, a novel adaptation scheme is proposed and is experimentally shown to be able to adapt to inaccurate statistics and changes of system states. Two static algorithms: Greedy and SA, have also been proposed for relatively static envrionments and for constructing initial trees under dynamic environments. Although we present our techniques in the context of streaming object dissemination, they can be generalized to other streaming data disseminatoin problems by revising the cost model.

References

[1] M. Aguilera, R. Strom, et al. Matching events in a content-based subscription system. In *PODC*, 1999.

[2] S. Babu, R. Motwani, et al. Adaptive ordering of pipelined stream filters. In *SIGMOD*, 2004.

[3] G. Banavar, et al. An efficient multicast protocol for content-based publish-subscribe systems. In *ICDCS*, 1999.

[4] S. Banerjee, et al. Construction of an efficient overlay multicast infrastructure for real-time applications. In *INFOCOM*, 2003.

[5] M. Blum, et al. The minimum latency problem. In *STOC*, 1994

[6] E. Brosh and Y. Shavitt. Approximation and heuristic algorithms for minimum delay application-layer multicast trees. In *INFOCOM*, 2004.

[7] A. Carzaniga, M. Rutherford, and A. Wolf. A routing scheme for content-based networking. In *INFOCOM*, 2004.

[8] Y. hua Chu, et al. A case for end system multicast. In *SIGMETRICS*, 2000.

[9] A. Carzaniga and A. Wolf. Forwarding in a content-based network. In *SIGCOMM'03*, 2003.

[10] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an internet-scale xml dissemination service. In *VLDB*, 2003.

[11] F. Fabret, et al. Filtering algorithms and implementation for very fast publish/subscribe. In *SIGMOD*, 2001.

[12] Y. E. Ioannidis and Y. Kang. Randomized algorithms for optimizing large join queries. In *SIGMOD*, 1990.

[13] D.S. Johnson, et al. Optimization by simulated annealing an experimental evaluation unpubished manuscript, 1987.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*

[15] S. Madden, et al. Continuously adaptive continuous queries over streams. In *SIGMOD*, 2002.

[16] S. Shah, et al. An efficient and resilient approach to filtering and disseminating streaming data. In *VLDB*, 2003.

[17] S. Shah, et al. Maintaining coherency of dynamic data in cooperating repositories. In *VLDB*, 2002.

[18] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Proceedings of the ACM/SIGCOMM Internet Measurement Conference (IMC-03)*, 2003.

[19] Y. Zhou, B. C. Ooi, K.-L. Tan, F. Yu. Adaptive Reorganization of Coherency-Preserving Dissemination Tree for streaming data. Technical report, 2005.