The VLDB Journal (2008) 17:1465–1483
DOI 10.1007/s00778-007-0077-7

REGULAR PAPER

Disseminating streaming data in a dynamic environment: an adaptive and cost-based approach

Yongluan Zhou · Beng Chin Ooi · Kian-Lee Tan

Received: 4 May 2006 / Revised: 20 January 2007 / Accepted: 9 August 2007 / Published online: 7 November 2007

Abstract In a distributed stream processing system, streaming data are continuously disseminated from the sources to the distributed processing servers. To enhance the dissemination efficiency, these servers are typically organized into one or more *dissemination trees*. In this paper, we focus on the problem of constructing dissemination trees to minimize the average *loss of fidelity* of the system. We observe that existing heuristic-based approaches can only explore a limited solution space and hence may lead to suboptimal solutions. On the contrary, we propose an adaptive and cost-based approach. Our cost model takes into account both the processing cost and the communication cost. Furthermore, as a distributed stream processing system is vulnerable to inaccurate statistics, runtime fluctuations of data characteristics, server workloads, and network conditions, we have designed our scheme to be adaptive to these situations: an operational dissemination tree may be incrementally transformed to a more cost-effective one. Our adaptive strategy employs distributed decisions made by the distributed servers independently based on localized statistics collected by each server at runtime. For a relatively static environment, we also propose two static tree construction algorithms relying on apriori system statistics. These static trees can also be used as initial trees in a dynamic environment. We apply our schemes to both single- and multi-object dissemination. Our extensive 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.

Keywords Streaming data dissemination · Dissemination trees · Distributed stream processing

1 Introduction

Distributed stream processing has been gaining increasing research attentions in the recent years. In such a system, queries submitted by the clients (e.g., continuous queries monitoring the streams or ad hoc queries on the historical and current status) would be distributed to the various processing servers for processing. To evaluate the client queries, the streaming data have to be disseminated from the sources to the distributed servers. Due to the continuity and massiveness of the data, 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 stream dissemination system, where a data source continuously disseminates fast changing data objects (e.g., sensor data, stock prices and sport scores) to a number of stream processing servers. Clients submit queries 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. The servers are organized into one or more dissemination trees (with the data source being the root node) so

Y. Zhou (✉)
Distributed Information Systems Laboratory,
School of Computer and Communication Sciences, EPFL,
EPFL-IC-LSIR, Bâtiment BC, Station14,
1015 Lausanne, Switzerland
e-mail: yongluan.zhou@epfl.ch

B. C. Ooi · K.-L. Tan
School of Computing, National University of Singapore,
Singapore, Singapore
e-mail: ooibc@comp.nus.edu.sg

K.-L. Tan
e-mail: tankl@comp.nus.edu.sg

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*, which has been used in previous work [17,18]. It measures the portion of time that the values in the servers conform to their coherency requirements. While adopting this metric, we extend it to accommodate generic data divergence metrics, which will be further explained in Sect. 2.1. In general, 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 neighbouing 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 relatively static systems. These static trees can also be used as initial trees in a dynamic environment.
- We apply the above schemes to both single object dissemination and multiple object dissemination problems. For multiple object dissemination, we study two approaches: single-tree approach and multi-tree approach.
- 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.

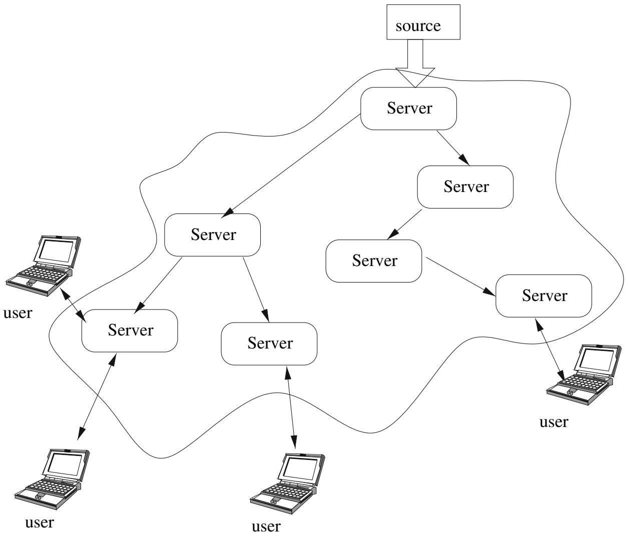


Fig. 1 The system architecture

A preliminary version of this paper appears in [21]. There, we present our scheme for single object dissemination. This paper extends the work in several ways. First, we extend the work to disseminate multiple objects. In particular, we propose two multiple object dissemination approaches. Second, we extend the cost model to incorporate the adaptation cost. Finally, we report results of additional performance study done to evaluate the adaptation overhead, as well as the multiple object dissemination strategies.

The rest of this paper is organized as follows. Section 2 presents the problem and motivations. In Sects. 3 and 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 Sect. 5. We review related work in Sect. 6. Finally, we conclude in Sect. 7.

2 Problem formulation and motivations

In this section, we first formulate the problem by presenting the system model, the definition of the metric as well as the formal problem statement. Then we motivate our work by identifying the potential problems of the existing techniques.

2.1 Problem formulation

Figure 1 shows the overview of the architecture of our system and Table 1 lists a number of major notations that would be used frequently throughout the whole paper. In the system, there is a data source s that stores a set of data objects $O = \{o_1, o_2, \ldots, o_{|O|}\}$, a set of servers $N = \{n_1, n_2, \ldots, n_{|N|}\}$, and a large number of clients. Each server n_i is a continuous

Table 1 Notations

s	The source node
N	The set of server nodes
n_i	The ith node
o_x	The xth data object
$LF(n_i)$	The lost of fidelity of n_i
C_i	The set of child nodes of n_i
T_i	The subtree rooted at n_i
GC_i	The set of grandchild nodes of n_i
O_i	The set of objects requested by n_i
O_i^m	The set of objects requested by the subtree rooted at n_i
$cr_{i,x}$	The coherency requirement of n_i on o_x
r_i	The rate of the update message meant for n_i
r_i^m	The rate of the update message meant for any node in the subtree rooted at n_i
r_i^c	The sum of the update rate over all nodes in the subtree rooted at n_i
$r_{i,x}$	The rate of the update message from o_x meant for n_i
$r_{i,x}^m$	The rate of the update message from o_x meant for any node in the subtree rooted at n_i
$d(n_i, n_j)$	The communication delay between n_i and n_j
$D(s, n_i)$	The total communication delay of the path from s to n_i in the tree
t_i^p	The time needed to perform filtering of a message at n_i
t_i^c	The time needed to perform transmission of a message at n_i
t_i^e	The time needed to collect information at n_i
t_i^d	The time needed to compute the adaptation benefit of a transformation at n_i
t_i^a	The amortized adaptation cost at n_i
t_i	The expected processing time at n_i
$g(n_i)$	The processing delay of message in node n_i
$p(n_i)$	The parent node of n_i
ρ_i	The workload of n_i

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 divergence of the data value from its exact value. Our approach does not restrict the way in which the divergence is measured. The possible metrics include the number of changes since the last update, the deviation of the values (for numerical data), the edit distance (for string data) or the difference of the update time stamp. Instead of just picking anyone of them, our system allows a customizable divergence function. We denote the divergence function as $\mathrm{DIV}(o_x(n_i, t), o_x(n_j, t))$, where $o_x(n_i, t)$ is the version of data object o_x cached in node n_i at time t.

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 $\mathrm{cr}_{i,x}$ of n_i on object o_x is equal to the most stringent requirement of its queries that involve o_x. To determine whether an update message should be transferred to the child node or a client, our system also employs a customizable function $match(m, n_i)$, which returns either $true$ or $false$ for a given message and a child node n_i. An application developer can design different functions for different divergence functions. Shah [18] proposed such a function for numerical data dissemination. We will not go into detail of the design of this function and concentrate on the construction and adaptation of dissemination trees in this paper.

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 $\mathrm{cr}_{i,x}^m$ on an object o_x which corresponds to the most stringent one among all the $\mathrm{cr}_{i,x}$s of the subtree rooted at n_i. Again, a customizable function is used to generate the $\mathrm{cr}_{i,x}^m$. A parent performs the filtering of messages based on the $\mathrm{cr}_{i,x}^m$ 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 [17,18], 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 $\mathrm{cr}_{i,x}$. Then the fidelity of n_i on data object o_x at time t is defined as:

$$f(n_i, o_x, t) = \begin{cases} 1 & : \ \mathrm{DIV}(o_x(s,t), o_x(n_i,t)) < \mathrm{cr}_{i,x} \\ 0 & : \ \mathrm{DIV}(o_x(s,t), o_x(n_i,t)) \geq \mathrm{cr}_{i,x}. \end{cases} \quad (1)$$

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 $\mathrm{LF}(n_i) = 1 - F(n_i)$. Our objective is to minimize the average loss of fidelity over all nodes

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

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* $\mathrm{cr}_{i,x}$ *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 trees 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 more restrictive problem is already NP-Hard [5].

2.2 Motivation

In view of the complexity of the problem, existing approaches such as DiTA [17] 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 only explore a limited solution space and 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 long distance from the source, 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 would suffer from the long processing delay in the slow server or the long transmission delay. This would result in severe loss of fidelity. Furthermore, multiple runs of trial and error is required to obtain an optimal fanout constraint. This may impede the deployment of the system. To handle these limits and find out the trade-offs, 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 is 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 sub-optimal. The problem of adapting to inaccurate statistics and system changes has been extensively explored in other problems such as query processing [2, 15]. 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 Single object dissemination

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

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 would 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 queuing time and the processing time. Let us estimate them one by one.

1. Queueing time. In our model, each node is a queuing system. From basic queuing theory, the expected queuing 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 processing time t. Hence to estimate the queuing 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 cr 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, i.e., $r_i^m = \max\{r_j | n_j \in T_i\}$. 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

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

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:

$$t_k = \sum_{n_j \in C_k} t_{kj}. \tag{3}$$

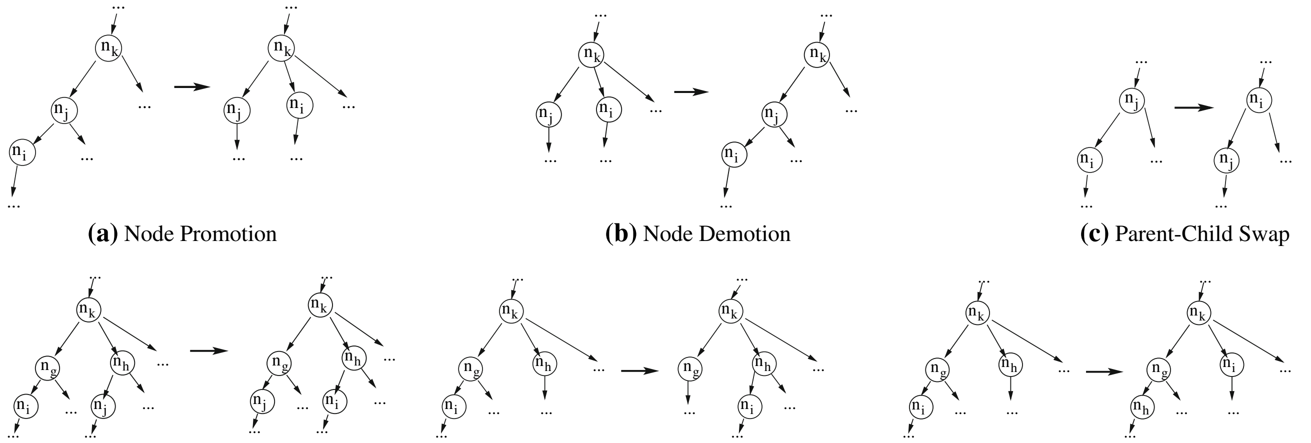
Given t_k, the average processing time of a message, we can derive that the workload of n_k is $\rho_k = r_k^m t_k$. Hence the queuing time of a message in node n_k is $\frac{\rho_k}{1-\rho_k}t_k$. Note that this covers both the queuing times for processing and transferring a message.

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

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

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. We can derive, with simple calculations, that this



(a) Node Promotion **(b)** Node Demotion **(c)** Parent-Child Swap

(d) Cousin Swap **(e)** Nephew Adoption **(f)** Uncle-Nephew Swap

Fig. 2 Local transformation rules

processing delay is

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

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

$$LF(n_i) = r_i \times [D(s, n_i) + g(p(n_i)) + g(p(p(n_i))) + \cdots + g(s)] \quad (6)$$

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. Neighboring states are generated based on a set of transformation rules. 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.

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.

1. **Node promotion**: Promote a node n_i to its parent's sibling. All the nodes in the sub-tree rooted at n_i are also moved along with n_i. Figure 2a 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. The children of n_i would also be moved along with n_i. In the example shown in Fig. 2b, 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. Again all their other children would be brought along with them. In Fig. 2c, 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. Their original children

would still be connected with them. Figure 2d 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 Fig. 2e, n_i's grandparent is the parent of n_h. In this transformation, n_i is added as a child of n_h. The children of n_i are moved along with it.

6. **Uncle–nephew swap:** Swap the positions of n_h with its nephew n_i. Again, their children are moved along with them. Figure 2f 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 presented above are proposed based on this intuition. While the composite transformations can be extended to involve arbitrary nodes, we only consider these transformations to keep the runtime adaptation scheme 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.

As depicted in Fig. 2a, 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

$$\Delta\text{AvgLF}_1 = \frac{1}{|N|} r_i^c \times \left[d(n_k, n_i) - d(n_k, n_j) - d(n_j, n_i) - g(n_j)\right],$$

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

$$\Delta\text{AvgLF}_2 = \frac{1}{|N|} \Big\{ (r_j^c - r_j)[g'(n_j) - g(n_j)] + (r_k^c - r_k)[g'(n_k) - g(n_k)] \Big\},$$

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. ΔAvgLF is equal to the sum of ΔAvgLF_1 and ΔAvgLF_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 may 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 to try all kinds of transformations, we restrict each node to consider only 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

Algorithm 1: AdaptationAttempt

```
begin
    $maxBenefit \leftarrow 0$; $t \leftarrow NULL$;
    for each possible transformations $t1$ involving the children
    and grandchildren do
        if $maxBenefit < Benefit(t1)$ then
            $maxBenefit \leftarrow Benefit(t1)$;
            $t \leftarrow t1$;
    if $t \neq NULL$ 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
```

moving a grandchild from one child to another child (nephew adoption). A node sends reorganization requests (if any) to the involved descendants, e.g., n_j in both Fig. 2a, b, n_i and n_j in Fig. 2c, n_g and n_h in both Fig. 2d, e, n_g in Fig. 2f. 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 be 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 when it is already holding a token, then it would also ignore the reorganization request to avoid any contradictions.

In the midst of a tree transformation, data are disseminated through the old path. After the new connections are created, the old connections are dropped and the dissemination is transferred to the new connections.

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 grand-children. 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 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 grandchildren. This information is collected only once and need not be collected again at runtime. This is 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.

3.2.4 Modeling the adaptation cost

The adaptation scheme incurs runtime overhead, which includes the cost of information collection and decision making and depends on the fanout of the nodes. To keep the adaptation cost low, there are two approaches: (1) extend the cost model to reflect the adaptation cost so that the tree construction would inherently restrict the fanout; (2) adopt a coarser-grained cost model when fanout increases. We study the first approach in this paper and defer the second one as our future work. Let the set of grandchildren of n_k be GC_k. Assume the time spent by n_k to collect information for one node be t_k^e and the time to consider each possible decision be t_k^d. Furthermore, the length of the information collection and decision making period be T_e and T_d, respectively. To estimate the adaptation overhead, we should estimate the number of nodes to collect information and the number of possible decisions to be considered. Obviously, the former number is equal to $|C_k| + |GC_k|$. To compute the latter one, we should add up the possibilities of each transformation rules. For example, in node promotion, there are $|GC_k|$ possible nodes to promote. In node demotion, each child node can be considered for demotion to the child of $(|C_k| - 1)$ nodes and hence there are $|C_k| \cdot (|C_k| - 1)$ possibilities. We can perform similar analysis on other transformation rules and the amortized adaptation cost t_k^a can be estimated as:

$$\begin{aligned} t_k^a = {} & \frac{t_k^e}{T_e}(|C_k| + |GC_k|) + \frac{t_k^d}{T_d}(|GC_k| + |C_k| \cdot (|C_k| - 1) \\ & + \sum_{n_j \in C_k} |C_j| \cdot (|GC_k| - |C_j|) \\ & + 2 \sum_{n_j \in C_k} (|GC_k| - |C_j|)). \end{aligned} \tag{7}$$

This cost can also be computed in constant time by storing and performing incremental updates of some of the intermediate values. t_k^a is added to $g(n_k)$ to extend our cost model to factor in the adaptation overhead.

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 Eqs. (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. For brevity, 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 $T$;
    $N[0] \leftarrow s$;
    $N[1 \cdots |N|-1] \leftarrow$ 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
        $e \leftarrow \arg\min_{0 \le j < i} AvgLF(T \cup \{N[i], e(n_j, n_i)\})$;
        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*

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

Proof Let us first look at the worst case communication delay of the messages sent to a node n_i. Because of the triangle inequality, when n_i is added to T, the transfer delay $d(n_k, n_i)$ between the parent n_k and n_i is less than $d(s, n_k) + d(s, n_i)$. Because the nodes are added to T in ascending order of d_i, we can get $d(s, n_k) + t_k^p + t_k^c < d(s, n_i) + t_i^p + t_i^c$ and hence $d(s, n_k) < d(s, n_i) + t_i^p + t_i^c$, i.e. $d(s, n_k) < d_i$. We can obtain the following expression:

$$\left.\begin{array}{l} d(n_k, n_i) < d(s, n_k) + d(s, n_i) \\ d(s, n_k) < d_i \\ d(s, n_i) < d_i \end{array}\right\} \Rightarrow d(n_k, n_i) < 2d_i.$$

We can also derive that the transfer delay of each edge in the path from the root to n_i is at most $2d_i$. Because the height of the tree is at most h, then the number of edges in the path from the root to n_i is at most h. That means the worst case communication delay for n_i is $2d_i \cdot h$.

Now we look at the worst case processing delay of messages sent to n_i. Since $t_k^p + t_k^c < d_i$, we have $t_k < C(t_k^p + t_k^c) < C \cdot d_i$ [from Eqs. (3), (2)). Furthermore, from Eq. (4) we have the following:

$$g(n_k, n_i) = \frac{1 + r_k^m \cdot t_k}{2(1 - r_k^m \cdot t_k)} t_k + t_k^p + t_k^c - \frac{1}{2} t_{ki}$$
$$< \frac{1 + r \cdot C \cdot d_i}{2(1 - r \cdot C \cdot d_i)} C \cdot d_i + d_i. \quad (8)$$

This is the worst case processing delay in the parent n_k. Since for any ancestor n_j, $t_j^p + t_j^c < d_i$ is also true, in Eq. (8) is also applicable to n_j. Again, the number of ancestors of n_i is at most h. Hence we can derive that the worst case total processing delay of a message sent to n_i is at most h times the worst case processing delay in each ancestor of n_i. □

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 [11]. 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

Algorithm 3: Simulated Annealing

```
begin
    $S \leftarrow S_0$; $T \leftarrow T_0$; $minS \leftarrow S$;
    while $!frozen$ do
        while $!equilibrium$ do
            $newS \leftarrow RandomNeighbor(S)$;
            $\Delta C \leftarrow cost(newS) - cost(S)$;
            if $\Delta C \le 0$ then $S \leftarrow newS$;
            else $S \leftarrow newS$ with probability $e^{-\Delta C/T}$;
            if $cost(S) < cost(minS)$ then $minS \leftarrow S$;
        $T \leftarrow reduceTemp(T)$;
    return minS;
end
```

[1] If every non-leaf node has at least two children, then $h \le \log |N|$. In addition, some studies [20] have shown that violations of triangle inequality is not very frequent, which is only about $1.4 - 6.7\%$.

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 [13,11], we select the parameters as follows: (1) T_0: $2 \times cost(S_0)$; (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 Sect. 3.2.1. The cost of the new scheme can be computed using the incremental cost computation presented in Sect. 3.2.1. 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.

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 single-tree approach (to build one tree for multiple data objects), and (b) the multi-tree approach (to build one dissemination tree for each data object). In the following subsections, we will look into these two approaches in detail.

4.1 The single-tree approach

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. In this section, we first develop the cost model for this approach, and then present the dissemination tree construction scheme.

4.1.1 Cost model

The derivation process is similar to the single object case, except that we have to deal with more than one object. The delay of a message for a node n_i can still be divided into two parts: the transmission delay and the processing delay in the path from the root to n_i. The transmission delay is the same as the single object case which is $D(s, n_i)$. Before estimating the processing delay of a message in each node, we extend some of the above notations as follows. The message arrival rate of n_k from object o_x is $r_{k,x}$ and its corresponding effective update arrival rate is $r_{k,x}^m$. The sum of $r_{k,x}^m$ over all objects is denoted as $r_k^m = \sum_{o_x \in O_k^m} r_{k,x}^m$. We assume the expected per-child filtering time and the transmission time for a message in n_k is equal over all of the objects, which are still denoted as t_k^p and t_k^c, respectively.

Now we are ready to derive the cost function of the processing delay. Recall that the delay is equal to the sum of the queuing time and the processing time. For a message from object o_x, the expected processing time in n_k for a child n_j interested in o_x is

$$t_{kj,x} = t_k^p + t_k^c \frac{r_{j,x}^m}{r_{k,x}^m}.$$

Hence the total processing time of a message from object o_x would be

$$t_{k,x} = \sum_{n_j \in C_{k,x}} t_{kj,x}.$$

The average processing time of a message from all the objects in O_k^m is

$$t_k = \frac{\sum_{o_x \in O_k^m} r_{k,x}^m t_{k,x}}{r_k^m}.$$

Then the workload of n_k can be computed as $\rho_k = r_k^m \cdot t_k$. Therefore, the expected queuing time of a message would be $\frac{\rho_k}{1-\rho_k} t_k$. Similar to the analysis in the single object case, the message received by a child n_j has to experience an average processing time of $\frac{1}{2}(t_{k,x} - t_{kj,x}) + t_k^c + t_k^p$. Summing up the queuing time and the processing time, we have the expected processing delay in n_k of a message for one of its child n_j on object o_x:

$$g(n_k, n_j, o_x) = \frac{\rho_k}{1-\rho_k} t_k + \frac{1}{2}(t_{k,x} - t_{kj,x}) + t_k^c + t_k^p. \qquad (9)$$

In Eq. (9), the cost function distinguishes the processing cost on different objects. That means if the number of objects is large, the computational cost of our algorithm would be very large. Therefore, we provide an approximation on the cost model as follows. First, we approximate $t_{kj,x}$ for all values of x by using

$$t_{kj} = \frac{\sum_{o_x \in O_j^m} r_{k,x}^m t_{kj,x}}{r_k^m}.$$

Then we use t_k to approximate $t_{k,x}$. In this way, we can approximate Eq. (9) as follows:

$$\begin{aligned} g(n_k, n_j) &= \frac{\rho_k}{1-\rho_k} t_k + \frac{1}{2}(t_k - t_{kj}) + t_k^c + t_k^p \\ &= \frac{1+\rho_k}{2(1-\rho_k)} t_k + t_k^c + t_k^p - \frac{1}{2} t_{kj}. \qquad (10) \end{aligned}$$

Note that this equation is of the same form as Eq. (4) in the single object cost model. Similar to the approximation we have done in the single object case, which uses the average processing delay over all the children to approximate that of every child of n_k, we have:

$$g(n_k) = \frac{1+\rho_k}{2(1-\rho_k)} t_k + t_k^c + t_k^p - \frac{1}{2|C_k|} t_k. \tag{11}$$

Hence we can calculate the expected LF of n_i on object o_x, $\mathrm{LF}(n_i, o_k)$ and then the expected LF of n_i averaging over all its interesting objects, $\mathrm{LF}(n_i)$.

$$\begin{aligned}\mathrm{LF}(n_i) &= \frac{1}{|O_i|} \sum_{o_x \in O_i} LF(n_i, o_x) \\ &= u_i[D(s, n_i) + g(p(n_i)) + g(p(p(n_i))) \\ &\quad + \cdots + g(s)]\end{aligned} \tag{12}$$

where

$$u_i = \frac{\sum_{o_x \in O_i} r_{i,x}}{|O_i|}.$$

Furthermore, the adaptation cost can be incorporated by adding $t_k^a \cdot |O_k^m|$ to $g(n_k)$, where t_k^a can be computed using Eq. (7).

4.1.2 Dissemination tree construction

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. More specifically, in the list in Sect. 3.2.3, the 1st and 4th points remain unchanged, while the 2nd and 3rd points are revised as follows:

- The values of O_j^m, u_j^c, $r_{j,x}^m$, t_j^p and t_j^c of each of its children or grandchildren n_j for each object o_x in n_j's effective object set O_j^m.
- The value of u_i^c of node n_i, where u_i^c aggregated u_j values of all the nodes in the subtree T_i rooted at n_i, i.e., $u_i^c = \sum_{n_j \in T_i} u_j$.

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 object.

4.2 The multi-tree approach

In this approach, one dissemination tree is created for each data object, which is similar to DiTA. Each tree only covers those servers that are interested in the corresponding data object. By doing so, update messages of an object will not be routed through the uninterested nodes.

The operations in each node is similar to the single-tree approach. When an update message arrives, the node checks the children that are involved and forward the message if necessary. Therefore, the cost model is similar to the single-tree approach.

Furthermore, we can perform the adaptive transformation of each tree independently and concurrently. Unfortunately, these trees are not independent. Two trees are correlated through those nodes that appear in both of them. Hence the change of one tree may affect the other trees through their common nodes. In particular, when a node n_i is making its adaptation decision for a tree, one of its children n_j may be performing the adaptation in another tree. Hence n_i's decision may not be based on the right information. Simply sequencing the transformation of the trees would slow down the adaptation.

To solve this problem, extra mechanism has to be incorporated. In our scheme, each node has three possible states: IDLE, WAIT and HOLD. As in the single-tree approach, the root node of each tree generates the token which is passed around the tree in a top–down manner. Each node that receives the token, before making the adaptation decision, sends out a "hold" message to all its children and enters the WAIT state. A child node that receives a hold message will reply with an acknowledgement message and enter the HOLD state when possible. The parent node that receives all the acknowledgements from its children, will perform the adaptation as usual if and only if it is not in the HOLD state. The details of this mechanism are presented in Algorithms 4 and 5.

Algorithm 4: Process Message

```
begin
    while true do
        wait for a new message $msg$;
        HandleMsg($msg$);
        if $state = IDLE || WAIT$ then
            for each $wait \in Q_{ready}$ do
                $wait \leftarrow Q_{ready}$.Dequeue();
                PerformAdapt($wait.tree$);
                send a token message to each node in
                $Child[wait.tree]$;
                continue;
            for each $msg$ in $Q_{token}$ do
                remove $msg$ from $Q_{token}$;
                HandleMsg($msg$);
            for each $msg$ in $Q_{hold}$ do
                remove $msg$ from $Q_{hold}$;
                HandleMsg($msg$);
                if $state = HOLD$ then break;
end
```

Algorithm 5: Helper Functions

```
1  Function HandleMsg($msg$)
2  begin
3      switch $msg.type$ do
4          case $HOLD\_MSG$
5              if $state = HOLD$ then $Q_{hold}$.Enqueue($msg$);
6              else if $state = WAIT$ then
7                  if $msg.num > NUM$ then
8                      PerformHold($msg.tree$)
                       /*deadlock prevention */
9                  else $Q_{hold}$.Enqueue($msg$);
10             else if $state = IDLE$ then
11                 PerformHold($msg.tree$)
12         case $TOKEN\_MSG$
13             if $state = HOLD$ then
14                 if $msg.tree = hold.tree$ then $state \leftarrow WAIT$
                   /* this token unlocks the hold state */
15                 else $Q_{token}$.Enqueue($msg$) /*put it in the
                   token queue */
16             if $state = IDLE || WAIT$ then
17                 if $\exists wait, wait.tree = msg.tree$ then break
                   /* ignore this msg */
18                 create a new object $wait$ and put it into $waitPool$;
19                 $wait.tree \leftarrow msg.tree$ /* initialize the $wait$
                   object */
20                 $wait.count \leftarrow Child[msg.tree].length$;
21                 $state \leftarrow WAIT$;
22                 create a new hold message $hmsg$;
23                 $hmsg.num \leftarrow NUM$;
24                 send one copy of $hmsg$ to each node in
                   $Child[msg.tree]$;
25         case $ACK\_MSG$
26             Look up $wait$ in $waitPool$ s.t.
               $wait.tree = msg.tree$;
27             $wait.count - -$;
28             if $wait.count = 0$ then
29                 $waitPool$.Remove($wait$);
30                 if $state = WAIT || IDLE$ then
31                     PerformAdapt($wait.tree$);
32                 else $Q_{ready}$.Enqueue($wait$);
33 end
34 Function PerformAdapt($tree$)
35 begin
36     perform adaptation of $tree$;
37     if $waitPool = \phi$ then $state \leftarrow IDLE$;
38     else $state \leftarrow WAIT$;
39 end
40 Function PerformHold($tree$)
41 begin
42     send an ack message to $msg.source$;
43     $hold.tree \leftarrow tree$;
44     $state \leftarrow HOLD$;
45 end
```

Note that without careful considerations, the above algorithm may incur deadlock. Consider two nodes n_i and n_j. n_i is the parent of n_j in one tree while it is the child of n_j in another tree. It is possible that n_i and n_j will send a "hold" message to each other at about the same time. If they keep waiting for acknowledgement from each other, deadlock occurs. Furthermore, they should not both enter the HOLD state. To solve the deadlock problem, we assign a unique integer number NUM to each node, which is implemented by using the unique IP address of every node. When a node in the WAIT state receives a hold message, it enters the HOLD state only when its number is smaller than that of the hold message's origin. Lines 6–9 in Algorithm 5 implement this scheme.

Now let us analyze the effectiveness of our algorithm in solving the distributed deadlock problem. First, to model the problem, a directed graph, called a parent–child graph (or P–C graph), can be generated, where a vertex represents a network node and a directed edge from n_i to n_j represents the fact that n_j is a child of n_i in at least one dissemination tree. Moreover, without any deadlock prevention scheme, a deadlock would happen if there is a cycle, $n_{i_1} \rightarrow n_{i_2} \rightarrow \cdots \rightarrow n_{i_p} \rightarrow n_{i_1}$, in the P–C graph and each node in the cycle is kept waiting for the acknowledgement from its immediate next node, i.e., n_{i_1} waits for n_{i_2}, n_{i_2} waits for n_{i_3} and so on. By using our proposed scheme, we have the following theorem:

Theorem 2 *The system is deadlock-free.*

Proof Without loss of generality, assume there is a cycle $n_{i_1} \rightarrow n_{i_2} \rightarrow \cdots \rightarrow n_{i_p} \rightarrow n_{i_1}$ in the P-C graph. If a deadlock happens in this cycle, then $\text{NUM}_1 < \text{NUM}_2 < \cdots < \text{NUM}_p < \text{NUM}_1$ has to be satisfied, where NUM_j is the NUM value of node n_{i_j}. Otherwise, if say $\text{NUM}_1 > \text{NUM}_2$ (note that NUM_j is unique so $\text{NUM}_1 \neq \text{NUM}_2$), then, when n_{i_2} receives a hold message from n_{i_1}, n_{i_2} will enter the HOLD state and hence the dead lock will not happen. However $\text{NUM}_1 < \text{NUM}_2 < \cdots < \text{NUM}_p < \text{NUM}_1$ would not be true at anytime. Therefore, deadlock will not exist. □

In addition, when a node n_i is ready to perform adaptations, the workload statistics of a child n_j may have changed due to the adaptation of the other trees. In order to let n_i make decisions based on updated statistics, such statistics will piggyback onto the acknowledgement message sent to n_i. This includes the change of the number of n_j's children as well as the change of the update rates of its children.

5 Experiments

In this section, we present a performance study of the proposed techniques, and report our findings.

5.1 Experiment configurations

The simulator is implemented using ns-2, a popular discrete-event simulator for networking research. The topology is generated using a power-law topology generator: inet [12]. We generate a network topology with 3,500 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 20 ms.

The expected filtering time and transmission time of each node is derived by using two respective uniform distributions. In our basic configuration, we set the average values of these times as 5 and 1 ms, respectively (which may vary in our experiments), and set the minimum values as 1ms and 0.125 ms 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 capture the load of processing user queries in the servers.

In addition, the adaptation interval of our adaptive scheme is set to 200 s and the information update window is set to 50 s. These values are chosen such that the system would not be over reactive to short term variances in our experimental setup. With higher data volumes, these intervals could be set shorter. 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 2.8 GHz CPU. We also implemented the optimization algorithms and the adaptation functions in C to study their performance. The adaptation overhead would be studied and modeled in the experiments.

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

1. **DiTA** [17]. 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.) The servers are added to the trees one by one. A server can serve another server only when its coherency requirement is at least as stringent as that of the other. A server n_i is added to each tree for each of its requesting data objects. Heuristics are applied to ensure that the level of n_i is as small as possible and secondarily the communication delay between n_i and its parent is also as small as possible. However, since DiTA is a distributed algorithm, these heuristics cannot guarantee the above objective. Hence we use a centralized version of DiTA which has the guarantees. Note that this is 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 a node n_i, another node within the partial tree, which has the smallest communication delay to n_i and still has available fanout degree, is selected to act 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 1 s. In the experiments, we use 100 traces as our basic dataset which would be varied.

5.2 Adaptation cost

In this section, we study the cost of performing adaptations using our C implementation. 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 for both the single-tree and multi-tree approaches. To keep the adaptation cost affordable, we have to set an appropriate adaptation period T_d. For example, if we can afford 5% of the CPU time for adaptation, we can set the adaptation period of this testing node as shown in Fig. 3. For example, if this node serves 10,000 objects, we have to set the adaptation period larger than or equal to 12 s. Therefore, to keep the adaptation responsive, the number of objects served by each server and the number of children and grandchildren should be kept to a certain limit. Note that constructing the tree using our extended cost model inherently considers this effect. The cost of collecting information is analyzed similarly. In the following experiments, we set both t_k^d and t_k^e as 1 μs in the cost model and the simulation.

5.3 Single object dissemination

In this subsection, we examine the algorithms in a single object dissemination situation. We utilize the synthetic data-

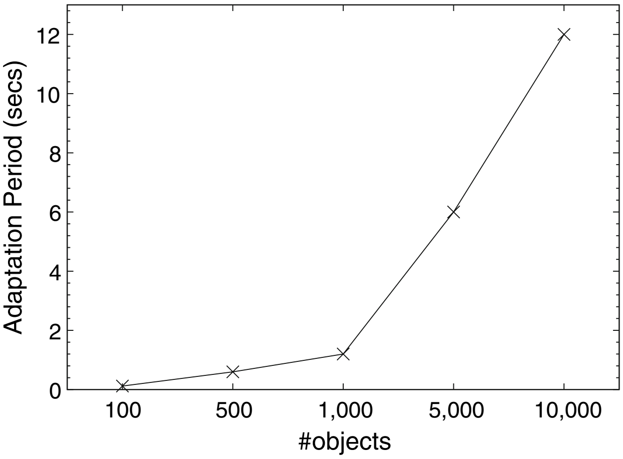


Fig. 3 Adaptation period selection

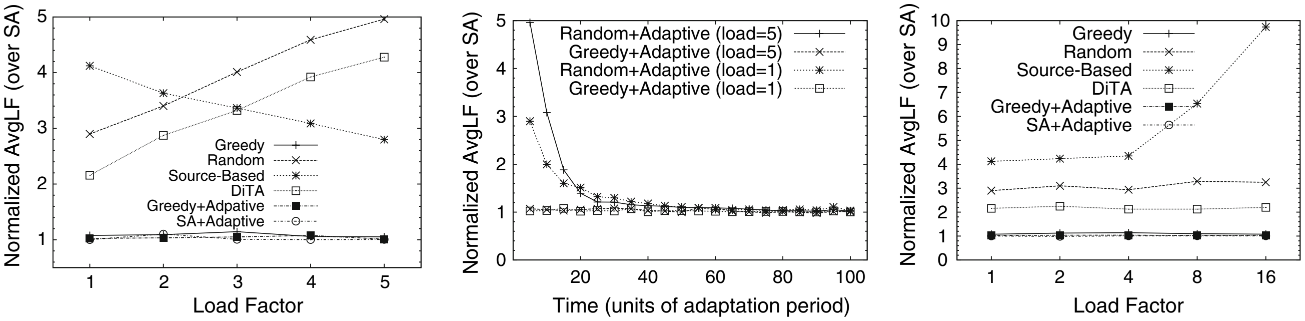
set. The expected message rate of each node is selected from a uniform distribution with the average value of 1 msgs/s and a minimum value of 0.5 msgs/s. (Note that the message rate models the coherency requirement at each node—a small coherency requirement implies a high message rate, and vice versa.)

5.3.1 Static environment

In the first experiment, we vary the average filtering time and transmission time by multiplying them with a parameter $load$. The parameter $load$ ranges from one to five 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 $20,000$ s and record the average AvgLF over the whole simulation period as well as the values within every $1,000$ s time window. To ease the comparison, we normalize the AvgLF values of all the other algorithms over that of the SA algorithm, which is (as expected) the best dissemination scheme.

Figure 4 shows the results of our experiment. From Fig. 4a, 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 Fig. 4a 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 schemes can identify



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

Fig. 4 Performance on single object dissemination in static environment

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 4b 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 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 Fig. 4c. 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 queuing 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.

5.3.2 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 perform worse than the others in this situation, we only examine the results of the other algorithms. We run the system for 20, 000 ys, and at the 10, 000th s, 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 ten 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.

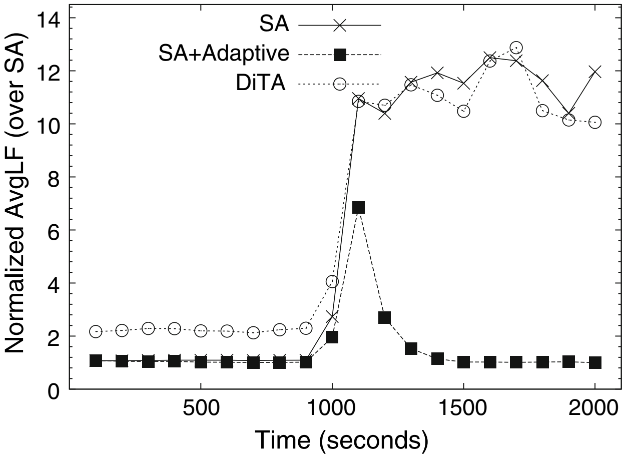
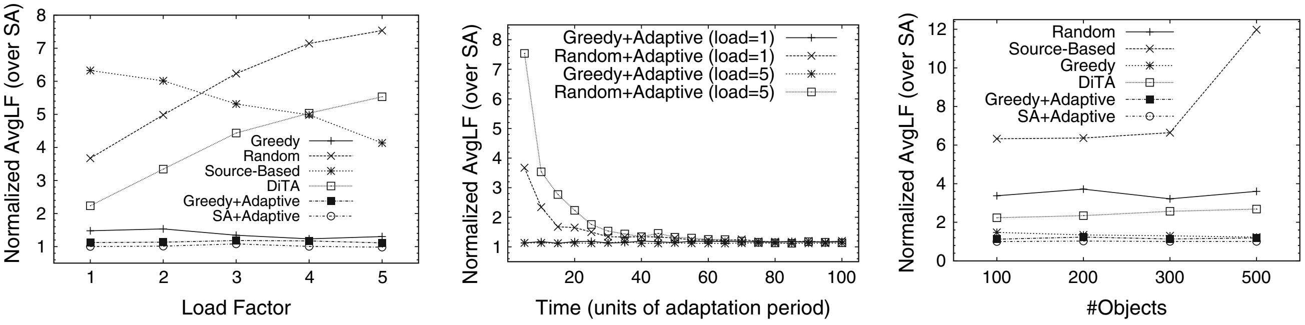


Fig. 5 Performance on single object dissemination in dynamic environment

The result is depicted in Fig. 5. In order to examine the optimality of the algorithms before and after the state transitions, we also executed two special runs of the SA algorithm: (a) run the SA algorithm based on statistics before the change. Let the AvgLF value of this run be SA1. (b) Run the SA algorithm based on statistics after the change. Let the AvgLF value of this run be SA2. We then normalized the AvgLF value of each algorithm under each condition by the corresponding AvgLF of the SA algorithm. For example, consider the DiTA scheme. Let the AvgLF be D. Then, before the change, its normalized value will be D/SA1, and after the change, its normalized value will be D/SA2. We compute the average of the normalized AvgLF values over a 1, 000 s window and then report the 20 resulting values. In Fig. 5, one can see that at the first 10, 000 s, SA and SA + Adaptive perform as well as SA, while DiTA is two times worse than them. After the 10, 000th s, the AvgLFs 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.



(a) Sensitivity to processing time **(b)** Online improvement **(c)** Sensitivity to number of objects

Fig. 6 Performance on multiple object dissemination

5.4 Multiple object dissemination

In the second set of experiments, we use our collected stock traces to examine the efficiency of our multiple object dissemination scheme. For each object, a probability that it is of interest to a server is set to 0.6, which will be varied in the experiments. The $cr_{i,x}$ values of each server n_i on each object o_x is chosen using a uniform random variable between 0.1 and 0.01. 100 traces are used as our basic configuration. For the ease of exposition, in the following experiments we first compare our single-tree approach with other approaches and then compare the single-tree approach with the multi-tree approach.

5.4.1 Single-tree approach

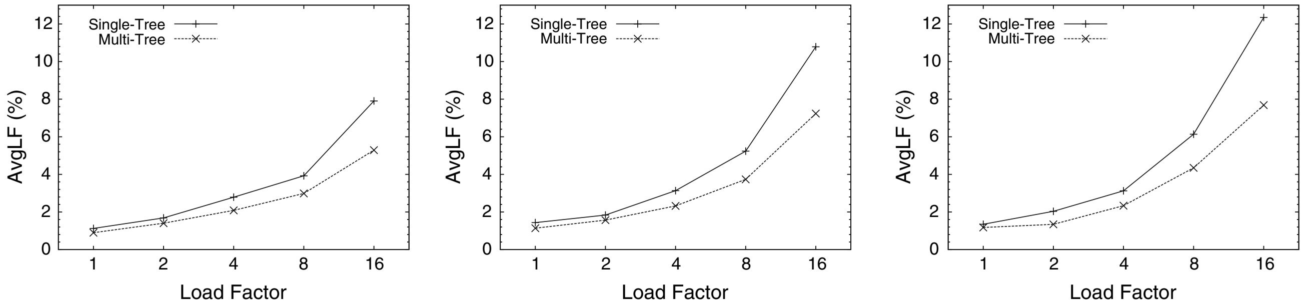
In the first experiment, we use a parameter *load* to vary the average filtering time and transmission time as we have done in the single object experiments. Figure 6a shows the results of this experiment. The relative performance of the algorithms is similar to the single object case. All our techniques perform as well as SA. Random and DiTA perform worse with larger *load* due to their inability to differentiate the capabilities of the various nodes. Source-based is insensitive to the parameter *load*. Figure 6b again shows that our adaptive mechanism can improve a random tree, which models a tree built on inaccurate statistics, to perform as well as SA.

In another experiment, we 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 Fig. 6c. With different number of data objects, Greedy, Greedy + Adaptive and SA + Adaptive consistently 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 AvgLFs 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 with respect to the number of objects.

5.4.2 Multi-tree approach

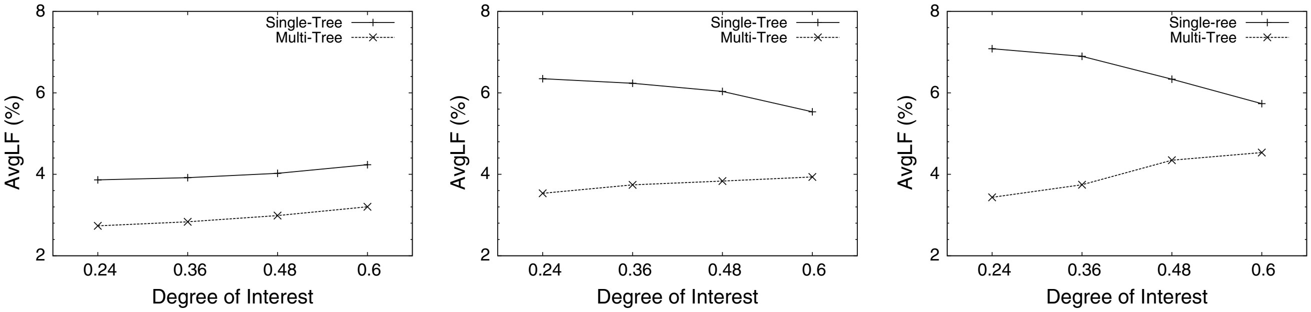
Now we study our multi-tree approach. From the results in the previous experiments, it is clear that our proposed single-tree method is superior to other methods. Thus, we shall only compare our multi-tree approach against our proposed single-tree method. Furthermore, for conciseness, only the results of SA for both approaches are presented. In the first experiment, we use a parameter *load* to vary the average filtering time and transmission time of the servers as we have done in Sects. 5.3.1 and 5.4.1. Figure 7 shows the result. It can be seen that the multi-tree approach outperforms the single-tree approach consistently. Furthermore, with higher server workload, their performance difference is larger. This is because the update messages in the multi-tree approach are transferred through fewer number of nodes and this benefit is more obvious with larger server load.

In the second experiment, we fix the *load* parameter at value eight. Instead, we vary the probability that a server is interested in an object for each pair of server and object. We refer to this probability as the degree of data interest. The smaller the degree of interest, the fewer are the number of objects of interest to each server. From the results shown in Fig. 8, it is obvious that the multi-tree approach consistently outperforms the single-tree approach. Moreover, when each server has a smaller number of interesting objects, we can achieve more benefit by using the multi-tree approach. The reason is the number of nodes in each individual dissemination tree is smaller and the update messages experience less processing delays in the servers. This effect is more obvious with a larger number of objects.



(a) 100 Objects **(b)** 300 Objects **(c)** 500 Objects

Fig. 7 Sensitivity on system workload



(a) 100 Objects **(b)** 300 Objects **(c)** 500 Objects

Fig. 8 Sensitivity on the number objects of interest to each server

5.4.3 Running time of SA and greedy

From the above experiment results, we can conclude that in a static environment, Greedy and SA perform the best among all the static algorithms given accurate statistics. In this experiment, we evaluate their running time. We use two sets of parameters of SA: (1) the parameters listed above and (2) changing $64 \times \#nodes$ to $16 \times \#nodes$ and $T < 0.001$ to $T < 0.0015$. Since the running time of the single-tree and the multi-tree approach is similar, only the results of the single-tree approach is presented here. Figure 9 shows the running time of both algorithms with different number of objects. Obviously, Greedy consistently outperforms SA in running time for both sets of parameters of SA. However, SA with parameters (2) comes with a plan whose cost is more than two times of that of Greedy. SA with parameters (1) can derive the best plan; however, the running time is significantly increased. We also tested a lot of other parameters of SA and cannot find a case that SA outperforms Greedy both in runtime and tree cost. For a static environment, SA is superior to Greedy due to its ability and robustness to find a low cost scheme. However, Greedy is more suitable for a dynamic environment, because it provides a cheaper way to construct a good initial tree and devoting more time to construct the initial tree does not make much sense as a previously optimal plan would become sub-optimal when the system state is changed. We can see this effect in Sect. 5.3.2 and the next section.

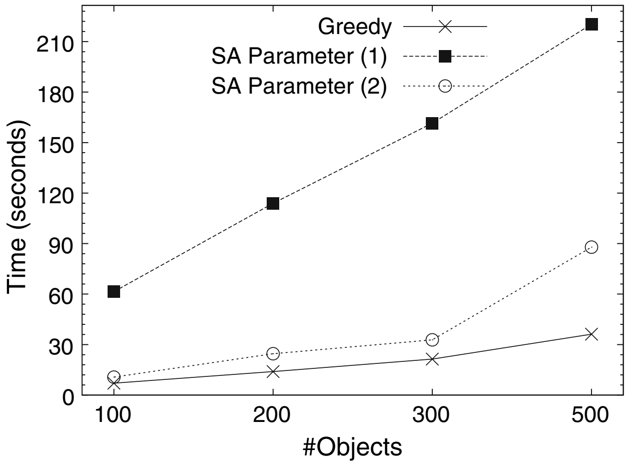


Fig. 9 Running time of greedy and SA

5.4.4 Dynamic environment

This experiment is similar to the one in Sect. 5.3.2, except that it is performed on multiple object dissemination. Since DiTA builds one tree for each object and DiTA has been shown above that it is not adaptable to system changes for

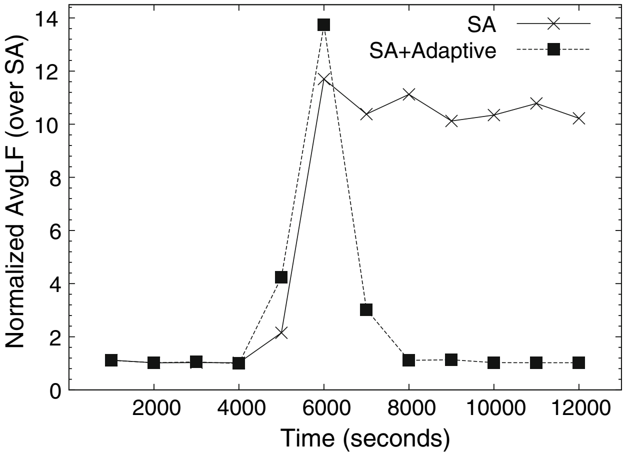


Fig. 10 Performance on multi-object dissemination in dynamic environment

any one of its dissemination trees, we only compare the SA and SA + Adaptive in this experiment. The other settings are similar to Sect. 5.3.2. At the 5, 000th s, 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 Fig. 10. We can see that before the change, SA works slightly worse than Adaptive. At the 5, 000th s, both SA and SA + Adaptive increase in their AvgLFs. 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 poor 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 re-optimize the scheme incrementally.

6 Related work

In [17,18], the authors introduced the problem of disseminating streaming data to preserve their coherencies. Two techniques were proposed to construct a dissemination tree: LeLA (Level by Level Algorithm) [18] and DiTA (Data item at a Time Algorithm) [17]. 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. Moreover, adaptivity is not addressed in DiTA. In addition [17], also proposed some fine-tuning techniques to reduce the system loss of fidelity. However, they are focused on divergence metrics that are measured by the deviation of numerical data values. Hence they cannot be applied in our system, where a generic divergence function is allowed.

The recently proposed application-layer multicast is shown to be much easier to deploy than IP layer multicast with only little penalties in performance [9]. 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 these techniques 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 [1,3,7,8]. However, most of these efforts focused on how to efficiently filter and route contents to the clients based on the clients' interests. They assume that there is an efficient scheme to organize the distributed dissemination servers and employ the schemes of general multicast systems. More recently, authors in [16] proposed a scheme to assign the dissemination servers to different dissemination channels based on the containment relationships of the user profiles. However, this paper also does not focus on dissemination tree construction algorithms and does not address the coherency problems.

7 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 environments and for constructing initial trees under dynamic environments. The Greedy algorithm is useful for dynamic environments due to its faster speed to build a relatively good initial tree, while SA is superior for static environments due to its robustness. Furthermore, the multi-tree approach is shown to be more robust to the number of objects, the degree of data interest as well as system workload.

There are several directions which we would like to explore further. First, although we have presented our techniques in the context of streaming object dissemination, they can be generalized to other streaming data dissemination problems by revising the cost model. In particular, we plan to examine how to generalize the system to support more complex queries such as those in relational databases. Second, we plan to study approximate cost model to keep the adaptation cost low. Finally, we also plan to study how failures of nodes can be handled gracefully. While some work has been done in this direction (e.g., [19]), these solutions are not designed for correlated failures (where multiple nodes fail at the same time).

Acknowledgments We would like to thank Feng Yu for his contributions to the initial implementation of the simulation. This work is partially supported by a university research grant R-252-000-237-112.

References

1. Aguilera, M.K., Strom, R.E., Sturman, D.C., Astley, M., Chandra, T.D.: Matching events in a content-based subscription system. In: PODC, pp. 53–61 (1999)
2. Babu, S., Motwani, R., Munagala, K., Nishizawa, I., Widom, J.: Adaptive ordering of pipelined stream filters. In: SIGMOD Conference, pp. 407–418 (2004)
3. Banavar, G., Chandra, T.D., Mukherjee, B., Nagarajarao, J., Strom, R.E., Sturman, D.C.: An efficient multicast protocol for content-based publish-subscribe systems. In: ICDCS, pp. 262–272 (1999)
4. Banerjee, S., Kommareddy, C., Kar, K., Bhattacharjee, S., Khuller, S.: Construction of an efficient overlay multicast infrastructure for real-time applications. In: INFOCOM (2003)
5. Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, W.R., Raghavan, P., Sudan, M.: The minimum latency problem. In: STOC, pp. 163–171 (1994)
6. Brosh, E., Shavitt, Y.: Approximation and heuristic algorithms for minimum delay application-layer multicast trees. In: IEEE INFOCOM'04 (2004)
7. Carzaniga, A., Rutherford, M.J., Wolf, A.L.: A routing scheme for content-based networking. In: INFOCOM (2004)
8. Carzaniga, A., Wolf, A.L.: Forwarding in a content-based network. In: SIGCOMM, pp. 163–174 (2003)
9. Chu, Y.H., Rao, S.G., Zhang, H.: A case for end system multicast. In: SIGMETRICS, pp. 1–12 (2000)
10. Diao, Y., Rizvi, S., Franklin, M.J.: Towards an internet-scale xml dissemination service. In: VLDB, pp. 612–623 (2004)
11. Ioannidis, Y.E., Kang, Y.C.: Randomized algorithms for optimizing large join queries. In: SIGMOD Conference, pp. 312–321 (1990)
12. Jin, C., Chen, Q., Jamin, S.: Inet: Internet topology generator. Technical Report CSE-TR-433-00, University of Michigan at Ann Arbor (2000)
13. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation. Part I, graph partitioning. Oper. Res. **37**(6), 865–892 (1989)
14. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
15. Madden, S., Shah, M.A., Hellerstein, J.M., Raman, V.: Continuously adaptive continuous queries over streams. In: SIGMOD Conference, pp. 49–60 (2002)
16. Papaemmanouil, O., Çetintemel, U.: SemCast: Semantic multicast for content-based data dissemination. In: ICDE, pp. 242–253 (2005)
17. Shah, S., Dharmarajan, S., Ramamritham, K.: An efficient and resilient approach to filtering and disseminating streaming data. In: VLDB, pp. 57–68 (2003)
18. Shah, S., Ramamritham, K., Shenoy, P.J.: Maintaining coherency of dynamic data in cooperating repositories. In: VLDB, pp. 526–537 (2002)
19. Shah, S., Ramamritham, K., Shenoy, P.J.: Resilient and coherence preserving dissemination of dynamic data using cooperating peers. IEEE Trans. Knowl. Data Eng. **16**(7), 799–812 (2004)
20. Tang, L., Crovella, M.: Virtual landmarks for the internet. In: Internet Measurement Comference, pp. 143–152 (2003)
21. Zhou, Y., Ooi, B.C., Tan, K.L., Yu, F.: Adaptive reorganization of coherency-preserving dissemination tree for streaming data. In: ICDE, p. 55 (2006)