Maintaining Coherency of Dynamic Data in Cooperating Repositories

Shetal Shah Krithi Ramamritham Prashant Shenoy†

TCS Lab for Internet Research,
Department of Computer Science and Engineering,
Indian Institute of Technology Bombay,
Mumbai, India 400076.

†Dept of Computer Science,
University of Massachusetts,
Amherst, MA 01003, USA.

Abstract

In this paper, we consider techniques for disseminating dynamic data—such as stock prices and real-time weather information—from sources to a set of repositories. We focus on the problem of maintaining coherency of dynamic data items in a network of cooperating repositories. We show that *cooperation* among repositories—where each repository pushes updates of data items to other repositories—helps reduce system-wide communication and computation overheads for coherency maintenance. However, contrary to intuition, we also show that increasing the degree of cooperation beyond a certain point can, in fact, be *detrimental* to the goal of maintaining coherency at low communication and computational overheads. We present techniques (i) to derive the "optimal" degree of cooperation among repositories, (ii) to construct an efficient dissemination tree for propagating changes from sources to cooperating repositories, and (iii) to determine when to push an update from one repository to another for coherency maintenance. We evaluate the efficacy of our techniques using real-world traces of dynamically changing data items (specifically, stock prices) and show that careful dissemination of updates through a network of cooperating repositories can substantially lower the cost of coherency maintenance.

1 Introduction

On-line decision making often involves significant amount of time-varying data. Examples of time-varying data include financial information such as stock prices and currency exchange rates, real-time traffic and weather information, and data from industrial process control applications. Due to its time-varying nature, users accessing such data items need to be provided with up-to-date values of these items. The coherency requirements associated with a time-varying data item depends on the nature of the item and user tolerances. To illustrate, a user involved in exploiting exchange disparities in different markets or an online stock trader may impose stringent coherency requirements (e.g., the stock price should never be out-of-sync by more than one cent from the actual value) whereas a casual observer of currency exchange rate fluctuations or stock prices may be content with less stringent coherency requirement.

Sources of time-varying data are often known to become bottlenecks especially when serving rapidly changing data to a large number of users (e.g., on election nights or when serving hot stock values in a volatile market). If clients refresh values of time-varying data items directly from the source, then the computational load at the sources will be high and hence (a) delays will occur in the dissemination of updates to clients, resulting in loss of data coherency, and (b) scalability of the system will suffer. One technique to alleviate this bottleneck is to replicate data across multiple repositories and have clients access the repository that is best positioned to meet their data coherency requirement. Although such replication can reduce load on the sources, replication of time-varying data introduces new challenges. First, data at the repositories needs to be coherent with the source. Second, unless updates to the data are carefully disseminated from sources to repositories, the communication and computation overheads involved in such dissemination can themselves result in delays as well as scalability problems, further contributing to loss of data coherency.

In this paper, we examine techniques to maintain the coherency of time-varying data items at a set of repositories. Each repository is assumed to store a subset of the dynamic data items, each of which has a coherency requirement associated with it. A particular focus of our work is to investigate how repositories can cooper-

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

ate with one another and the source to reduce the overheads of coherency maintenance. To do so, we assume that repositories storing a particular data item are logically connected to form an overlay network that we refer to as a *dynamic data dissemination tree* (abbreviated as the d^3t). The source of the data item forms the root of the d^3t. Instead of directly disseminating changes to a data item to all interested repositories, the source only pushes these changes to its children in the d^3t (each child is also referred to as a *dependent* of its parent). Each repository in turn pushes these changes to its dependent repositories. Dissemination using the d^3t incurs two kinds of overheads:

1. *Communication delays:* This is the delay incurred in propagating an update from a repository to a dependent. It includes all communication related delays, including the message processing delays at the source and destination of a message and the delays on all physical links between the two.

2. *Computational delays:* This is the delay resulting from the computations performed by a repository to determine whether an incoming data change is to be forwarded to one of its dependents.

The objective is to construct a d^3t that reduces these overheads while meeting the coherency requirements at all repositories. We assume that each dynamic data item will have its own d^3t; the logical structure of this tree depends on the dynamics of the data item, the coherency needs of the repositories, node to node communication delays, and the computational delays at each repository.

Given such an architecture, we consider two key issues in this paper:

1. How should the repositories be interconnected so as to minimize the overheads of maintaining coherency of all data items stored in the various repositories?

2. When should a node (i.e., a source or a repository) push changes of interest to other repositories so as to meet coherency requirements of the data item at all repositories?

In the rest of this section, we first define the problem of maintaining coherency for a data item and then describe the challenges in addressing the problem in a network of cooperating repositories.

1.1 Data Coherency Semantics

Consider a user interested in time-varying data items. Assume that the user obtains these items from a data repository instead of the source. Further, assume that the user specifies a coherence requirement (c) for each item of interest. The value of the c denotes the maximum permissible deviation from the value at the source, and thus, constitutes the user-specified tolerance. The coherency requirements can be specified in units of *time* (e.g., the item should never be out-of-sync by more than 5 minutes) or *value* (e.g., a stock price should never be out-of-sync by more than ten cents). In this paper, we only consider coherence requirements specified in terms of the value of the object; maintaining coherence requirements in units of time is a simpler problem that requires less sophisticated techniques (e.g., push every 5 minutes). To maintain coherence, each data item in the repository must be refreshed in such a way that the user-specified coherency requirements are maintained. Formally, let $S_x(t)$ and $U_x(t)$ denote the value of a data item x at the source and the user, respectively, at time t (see Figure 1). Then, to maintain coherence , we should have

$$|U_x(t) - S_x(t)| \leq c$$

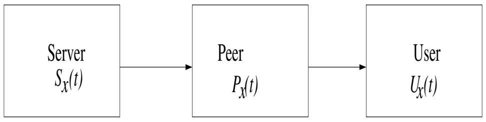


Figure 1: The Problem of Coherence

Although Figure 1 shows a single data repository, the coherency requirements are no different if there are multiple data repositories acting as intermediaries between the source and the end-user.

The effectiveness of such a cooperating repository can be quantified using a metric referred to as *fidelity*. The fidelity of a data item is the degree to which a its coherency requirements are met. We define fidelity f observed by a user to be the total length of time for which the above inequality holds normalized by the total length of the observations. The goal of a good coherency mechanism is to provide high fidelity at low cost.

1.2 Maintaining Data Coherency in the d^3t

Consider an architecture consisting of one or more sources, multiple repositories and several clients (see Figure 2). Each client in this architecture connects to one of the repositories to access dynamic data items; the choice of a particular repository depends on factors such as proximity, data availability, etc., and can be viewed as a separate problem. As indicated earlier, the client specifies a coherency requirement c for each data item of interest. Since multiple clients may be interested in the same data item, the coherency requirement for data item x at a repository P is defined to be the most stringent coherency requirements across all clients that obtain x from P. Consequently, depending on the needs of its clients, each repository can derive its own data needs and the associated coherency requirements. Hence, from now on, we focus on the source-repository coherency maintenance problem, that is, *our goal is to ensure that* $\forall t,\ |P_x(t) - S_x(t)| \leq c$ *where* $P_x(t)$ *and* $S_x(t)$ *denote the value of data item* x *at repository* P *and the source,* *respectively, and* c *is the coherency requirement of* x *at the repository.*

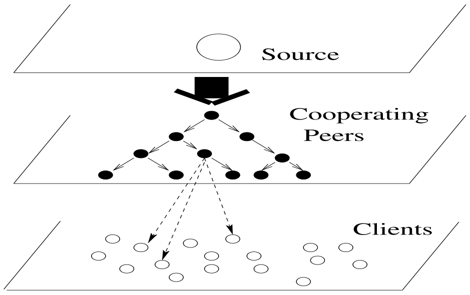


Figure 2: The Cooperative Repository Architecture

To maintain these coherency requirements, repositories are assumed to cooperate with one another. Such cooperation involves receiving updates to data items from a parent and pushing these updates to dependent repositories in the d^3t. The design of such techniques requires resolution of the following interrelated issues.

1. How much should a repository cooperate?

   Given that repositories cooperate with one another, a repository may hold data beyond what its own users may need. Further, cooperation requires a repository to expend both computation and communication resources to disseminate updates. *We show that this altruism pays off in reducing the system-wide overheads for maintaining coherency across all repositories. However, contrary to intuition, we also show that increasing the amount of cooperation beyond a certain point can, in fact, be* detrimental *to the overall goals of achieving high fidelity at low overheads.* To address this issue, we propose a technique to derive the "optimal" degree of cooperation among repositories.

2. Once the level of cooperation is decided, what should the (logical) interconnection between the repositories be, i.e., who serves whom?

   The structure of the d^3t determines the precise computational and communication costs at each repository. *We show that (i) the* d^3t *should be structured so as to balance these costs and (ii) so long as these costs are taken into account, the exact algorithm employed to construct the* d^3t *has only a minimal impact on the achieved fidelity.*

3. Given a d^3t, when should a repository disseminate updates (that it receives) to other repositories dependent on it?

   Since different repositories can have different coherence requirements, a repository will need to take these differences into account when disseminating updates to its dependents. We show that it is necessary to place repositories with stringent coherency requirements closer to the source. Also, a repository may have to receive more than the updates it itself needs so as to meet the coherency needs of its dependent—*even if the coherency needs of its dependent are less stringent than its own!*

In the following sections, we examine each question in turn, offer a set of solutions to address these questions, and demonstrate their efficacy using real-world traces of dynamically changing data.

The rest of the paper is structured as follows. We discuss details of our architecture for cooperating repositories in Section 2. Section 3 discusses techniques for determining the degree of cooperation, while Section 4 presents algorithms for constructing the d^3t. Techniques for disseminating updates from a node to its dependents are discusses in Section 5. Section 6 presents the results of our experimental evaluation. Section 7 presents related work, and finally, Section 8 presents our conclusions and directions for future work.

2 Architecture for Cooperating Repositories

Consider the cooperative repository architecture shown in Figure 2. We assume that the architecture uses the *push* approach for disseminating updates—the source pushes updates to its dependents in the d^3t, which in turn push these changes to their dependents and the end-clients. Not every update needs to be pushed to a dependent—*only those updates necessary to maintain the coherency requirements at a dependent need to be pushed.* To understand when a update should be pushed, let c^p and c^q denote the coherency requirements of data item x at repositories P and Q, respectively. Suppose P serves Q. To effectively disseminate updates to its dependents, the coherency requirement at a repository should be *at least as stringent as those of its dependents*:

$$c^p \quad \leq \quad c^q \tag{1}$$

Given the coherency requirement of each repository and assuming that the above condition holds for all nodes and their dependents in the d^3t, we now derive the condition that must be satisfied during the dissemination of updates. Let $x_i^s, x_{i+1}^s, x_{i+2}^s, \ldots x_{i+n}^s \ldots$ denote the sequence of updates to a data item x at the source S. Let $x_j^p, x_{j+1}^p, \ldots$ denote the sequence of updates received by a dependent repository P. Let x_j^p correspond to update x_i^s at the source and let x_{j+1}^p correspond to update x_{i+k}^s where $k \geq 1$. Then,
$\forall m = 1, k-1, |x_{i+m}^s - x_i^s| \leq c^p$
Thus, as long as the magnitude of the difference between last disseminated value and the current value is less than the coherency requirement, the current update is not disseminated (only updates that exceed the coherency tolerance c^p are disseminated). In other words, the repos-

itory P sees only a "projection" of the sequence of updates seen at the source. Generalizing, given a d^3t, each downstream repository sees only a projection of the update sequence seen by its predecessor.

In database terms, such a projection can be seen a "view" of the data stream. Maintaining this view for a data stream x involves ensuring that the projection of the data stream for a repository satisfies the coherency associated with x by that repository. This paper's contribution lies in developing techniques for efficiently maintaining the views of data streams at repositories according to the coherency specified by the repositories.

Efficient view maintenance techniques are required because, even if the all necessary updates are propagated by a repository to its dependents based on the condition developed above, due to the non-zero computational and communication delays in real-world networks and systems, data at a dependent will experience loss of coherency. Thus, it is impossible to achieve 100% fidelity in practice, even in expensive dedicated networks. The goal of our cooperative repository architecture is to achieve better fidelity in real-world settings where computational and communication overheads are non-negligible.

Finally, we note that the problem of dynamic data dissemination differs from both caching and replication in several significant ways. [20] lists these differences.

3 How Much Should a Repository Cooperate?

In this section and the next, we consider the issue of how to construct the d^3t. We define the degree of cooperation offered by a repository P for a data item x to be the maximum number of dependents that are served x by P. This is the fan out of the d^3t used for x. A high degree of offered cooperation implies that a repository is willing to take on increased responsibilities, which can help reduce source overload and potentially improve fidelity. On the other hand, a repository with a high degree of cooperation can also indirectly lead to a loss in fidelity (since this increases the computational overheads at a repository, which could become a bottleneck). Essentially, a repository that offers a high degree of cooperation may just transfer the source load onto itself.

Thus, a greater degree of cooperation increases the computational delay but reduces the end-to-end network delay (by virtue of reducing the path length from the source to the farthest repository). On the other hand, a small degree of cooperation reduces the computational delay at a repository but increases the end-to-end communication delays. In the extreme case, if the degree of cooperation is reduced to one, the d^3t becomes a linear chain of repositories with a large network delay. To maximize fidelity, the d^3t should be constructed such that the *sum of two delays components is minimized.*

As shown in Figure 3, for a given set of repositories,

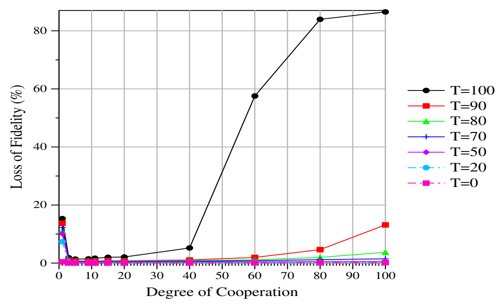


Figure 3: Need for Limiting Cooperation

the variation in (loss of) fidelity with increasing degree of cooperation exhibits a U-shaped curve. The left end of the x-axis corresponds to the d^3t being a chain and the right end to the case where a source directly disseminates updates to all its dependents. This curve portrays the results for different values of a parameter T—which encodes the stringency of the overall coherency requirements of repositories. (Section 6 presents details of how these curves were derived.) For now, it suffices to know that $T = 100\%$ signifies that all repositories have very stringent coherency requirements. As we can see from the plots, except when repositories do not have stringent coherency requirements, the choice of the degree of cooperation does make a difference on the achieved fidelity.

The point where the loss in fidelity is minimized depends on the minimum total network and computational delays incurred by the d^3t. In the falling part of the U-shaped curve, the communication delays dominate and in the rising part, the computational delays dominate. The figure shows that *arbitrarily increasing the degree of cooperation can, in fact, be detrimental to fidelity.* Hence, in a system where communication delays dominate, it is prudent to use a high degree of cooperation. On the other hand, if computational delays dominate, then a small degree of cooperation should be chosen (i.e., each repository should have a small number of dependents). In other words, the degree of cooperation should be directly proportional to the communication delays and inversely proportional to the computational delays. This results in the following heuristic to compute, $coop\_degree$, the degree of cooperation to be used:

$$\min\left(\frac{1}{C} \times \frac{average\ comm\_delay}{average\ comp\_delay}, \#cores\right) \quad (2)$$

where $\#cores$ denotes the upper bound on the available number of cooperative resources (i.e., maximum value of $coop\_degree$), $average\ comm\_delay$ and $average\ comp\_delay$ denote the average communication delays from one repository to another and the average computational delays in disseminating an update from one repository to all its dependents, respectively.

The above formula assumes that on average, only $C\%$ [1] of the dependents of a node would be interested in an update. This fraction is used to determine the effective computational delay for processing a data item at a node. Thus, the above formula allows us to set the degree of cooperation depending on the expected overheads. In Section 6 we study the effectiveness of this formula in setting the $coop\_degree$.

4 What Should the Logical Structure of Cooperating Repositories Be?

Consider a repository Q that needs to be inserted into the cooperative repository architecture. For simplicity, we describe the algorithm assuming the following scenario:

- When a repository Q wishes to enter the network it specifies, the list of data items of interest, their c values, and its degree of cooperation.
- A single source, S, can satisfy the needs of Q. (The extension to deal with multiple sources is fairly straightforward, we can run the algorithm once per source.)
- If a repository's data needs change or its data coherency needs change, then to handle the changed requirements, the algorithm is reapplied. Due to space constraints, we do not go into the details of this step in this paper.

Our algorithm constructs a single dynamic data dissemination graph d^3g during a single traversal of the repository network starting with the source S. For any particular data item x, the d^3g reduces to a tree (i.e., the d^3t) that consists of the paths along which an update to x is disseminated. Put another way, the d^3g is the union of all d^3t of data items of interest.

We call our algorithm LeLA (Level by Level Algorithm), because it looks for a position for Q in the current d^3g, level by level. S is at level 0, the repositories to which S disseminates data are in level 1, the dependents of repositories at level l are at level $l+1$, and so on.

The idea behind our algorithm is that starting at level 0, repositories at the current level are examined for their suitability to serve the new repository Q, that is, whether Q can become the dependent of a set of repositories at that level. This decision is made by a specially designated *load controller node* at each level. S vacuously serves as the load controller for level 0. The source examines if it can serve Q, if not it passes it on to the load-controller of the next level.

[1] We studied the sensitivity of our results to the constant C. We ran the experiments on different traces for 100 items each to determine the sensitivity of C to a chosen trace. Our results indicated that if the value of $C \geq 30$, then the resultant fidelity is high. (For our graphs $C = 30$ translates to a degree of cooperation around $9 - 10$ whereas $C = 100$ gives us a degree of cooperation around $3 - 4$. In general, the resulting fidelity is insensitive to the value of C if C is greater than 30. Variation in fidelity loss in such cases is only around 1%.

The function of the load controller at a level l is to find a set of suitable $parent$ repositories, P, at that level to serve the data and coherency needs for the new dependent repository, Q. For each repository in its level, the load controller calculates a *preference factor*. The smaller this factor, the more preferred a repository P is to be a parent of Q. We will explain the calculation of the preference factor shortly. For now, let us assume that this factor has been calculated for each repository at level l. To be a candidate for a parent, we consider all repositories whose preference values are within 5% of the smallest preference value computed for the current level.

Considering nodes whose preference factors are close to the smallest preference factor allows multiple repositories to become parents of Q, each serving a different subset of data needed by Q. A potential parent P can serve a data-item x to Q, if both Q and P are interested in x and if the coherency requirement of P for x is at least as stringent as that of Q. If more than one repository can serve a data-item x to Q then the more preferred among these is asked to serve x to Q.

It is quite likely that Q might want some data-items, say, $x_i, ..., x_k$, which are not served by any of the potential parents. The most preferred repository P is made to serve $x_i, ..., x_k$ to Q. This process of *augmenting* a parent's data requirements—to serve the needs of a new child—can have a cascading effect: For each of these data-items, P checks if any of its parents are serving it and if so, requests the parent for service, else it randomly selects one of its parents and asks it to service the data-item to P at the coherency required by Q. This is continued all the way up the d^3g till there is a path from the source to Q for those data-items.

The number of dependents currently served by a repository should be smaller than its $coop\_degree$. If a repository already has as many dependents as the $coop\_degree$ then it is not considered as a potential parent. As long as there are repositories with less dependents than the $coop\_degree$ specified, the load controller will find suitable parents from its level. If all of the repositories have reached their limit of $coop\_degree$ dependents, the load controller passes the request to the load controller of the next level.

The following factors are used to determine the preference factor of a node:

1. *Data Availability Factor:* The number of data items that a parent can serve Q, with its current data and coherency requirement.

2. *Computational delay Factor:* The larger the computational delay incurred at a parent P to disseminate a data change to its dependents, the less preferred it is. We approximate this delay by the number of dependents P has: On average, the more dependents P has, the greater will be the computational delays encountered by Q to get a data update from P.

3. *Communication delay Factor:* Parents which have a large communication delay with Q are less preferred.

Since we want to choose parents such that the delays are low and the data availability is high, we calculate the preference factor as: $\frac{delay(P,Q) \times numDependents(P)}{num\ data\ items\ P\ can\ serve\ Q}$

The load controller uses the above-mentioned preference factor to derive a preference value for each node at the current level and the ones with values within 5% of the minimum value are considered as potential parents. To this end, a load controller's view of a repository at its level is updated whenever a new repository becomes its dependent.

5 When Should an Update be Disseminated?

Assuming a d^3t has been constructed for data item x, consider a source S that disseminates x to a repository P, which in turn disseminates x to a dependent repository Q.

Recall from 1 that to effectively disseminate updates, we require that the coherency requirement at P should be at least as stringent as those of Q.

Let x^s_i, x^s_{i+1}, x^s_{i+2} ... denote a sequence of updates to x at the source S. Let x^p_j, x^p_{j+1}, x^p_{j+2} ... denote the updates received by P and x^q_k, x^q_{k+1}, x^q_{k+2} .... denote the updates received by Q. Since $c^p \leq c^q$, the set of updates received by Q is a subset of that received at P, which in turn is a subset of unique data values at the source. Specifically, an update x^p_j received by P is forwarded to Q if

$$| \, x^p_j - x^q_k \, | \quad \geq \quad c^q \tag{3}$$

where x^q_k denotes the previous update received by Q. Intuitively, Eq. (3) indicates that any update that violates the coherency requirements of Q is forwarded to Q.

We now show that this is a necessary but not sufficient condition for maintaining coherency at Q. Suppose x^s_i, x^p_j and x^q_k represent the value of x at S, P and Q, respectively. Let the next update at the S be x^s_{i+1} such that

$$| \, x^s_{i+1} - x^p_j \, | \quad < \quad c^p \tag{4}$$

$$| \, x^s_{i+1} - x^q_k \, | \quad \geq \quad c^q \tag{5}$$

Thus, the next update is of interest to repository Q but not to P. Since S is logically connected only to P, if S does not disseminate this update to P, then Q, a dependent of P, will also miss this update (causing a violation of its coherency requirement). Figure 4 provides an example of this situation. Thus, even under ideal conditions of zero processing and communication delays, a dissemination technique that solely uses Eq. (3) to disseminate updates might not provide 100% fidelity (indicating Eq. (3) is not a sufficient condition to maintain coherency). Hence, dissemination algorithms need to be developed carefully to avoid such *missed update* problems (i.e., should ensure that a repository does not miss any updates of interest to itself or its dependents).

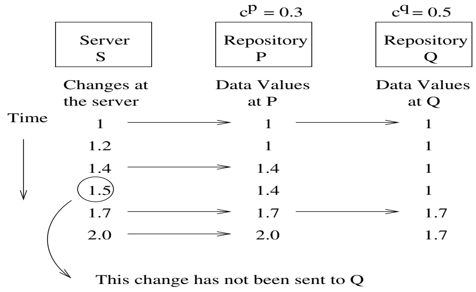


Figure 4: Need for Careful Dissemination of Changes

Next, we present two approaches to address this issue and also examine the entailed overheads.

5.1 Distributed (repository-based) Approach

The missed updates problem described earlier occurs when an update x^s_{i+1}, where $x^s_i < x^s_{i+1} < x^s_i + c^p$, satisfies both Eqs. (4) and (5).

From these equations, we get,

$$| \, x^s_{i+1} - x^p_j \, | - | \, x^s_{i+1} - x^q_k \, | \quad < \quad c^p - c^q \tag{6}$$

which reduces (as shown in [20]) to

$$c^q \quad - \quad | \, x^p_j - x^q_k \, | \quad < \quad c^p \tag{7}$$

Eq. (7) represents the additional condition that must be checked by any repository P to see if an update should be disseminated to its dependent Q. These two conditions can be proven to achieve 100% fidelity at all repositories (as sketched in [20]) Note that this applies even to the source, i.e., when P is the source. Thus, the dissemination technique propagates an update x^p_j received by P to dependent Q if either Eq. (3) or (7) is satisfied. In the example illustrated in Figure 4, such a technique would propagate the update corresponding to value 1.4 from P to Q (since it satisfies Eq. (7)). Consequently, the subsequent increase in value to 1.5 does not result in a c violation at Q. Note that the update of 1.4 is not strictly required as per the coherency requirement of Q (Eq. (3)), but is essential to prevent the "missed updates" problem.

5.2 Centralized (source-based) Approach

In this approach, the source maintains a list of all the unique coherency requirements for a data item x specified by various repositories. For each such coherency requirement, the source also tracks the last update disseminated for that coherency requirement. Upon a new update, the source examines each unique coherency requirement and the last update sent for that c. It then determines all c's that are violated by the update. The update is tagged by the maximum such coherency requirement c_max and the tagged update is then disseminated through the d^3t. The source also records this data value as the last update sent for all cs that are less then or equal to c_max.

Each repository receiving the update forwards it to all dependents that (i) are interested in the data item, and (ii) have a coherency requirement less than or equal to the tagged value. As sketched in [20], this dissemination algorithm also achieves a fidelity of 100% (in the absence of network transmission delays).

We now discuss the overheads of this approach. This algorithm finds the maximum coherency value, if any, affected by the an update at the source. A large network of cooperating repositories can result in a large overhead at the source (especially if the number of unique c values is also large). Since this approach disseminates updates only when necessary and only to repositories that need the update, the approach makes efficient use of the communication resources. The algorithm also imposes a state space overhead at the source to store the list of all unique coherency tolerances associated with each data-item and the last update sent for each c.

In summary due to the computational and space overheads, this approach may affect the scalability of the source compared to the distributed repository based dissemination approach. We study this issue in Section 6.

6 Experiments and Results

In this section, we demonstrate the efficacy of our techniques through an experimental evaluation. In what follows, we first present the experimental methodology and then the experimental results.

6.1 Experimental Methodology

Traces – Collection procedure and characteristics: The performance characteristics of our solution are investigated using real world stock price streams as exemplars of dynamic data. The presented results are based on stock price traces (i.e., history of stock prices) obtained by continuously polling *http://finance.yahoo.com* . We collected 100 traces making sure that the corresponding stocks did see some trading during that day. The details of some of the traces are listed in the table below to suggest the characteristics of the traces used. (*Max* and *Min* refer to the maximum and minimum stock prices observed in the 10000 values polled during the indicated *Time Interval* on the given $Date$ in Jan/Feb 2002.) As we can see, we were able to obtain a new data value approximately once per second. Since stock prices change at a slower rate than once per second, the traces can be considered to be "real-time" traces.

Company	*Date*	*Time Interval*	*Min*	*Max*
Microsoft	Feb 12	22:46-01:46 hours	60.09	60.85
SUNW	Feb 1	21:30-01:22 hours	10.60	10.99
DELL	Jan 30	00:43-04:12 hours	27.16	28.26
QCOM	Feb 12	22:46-01:46 hours	40.38	41.23
INTC	Jan 30	00:43-04:12 hours	33.66	34.239
Oracle	Feb 1	21:30-01:22 hours	16.51	17.10

Characteristics of some of the Traces used for the experiment

Repositories – Data, Coherency and Cooperation characteristics: We simulated the situation where all repositories accessed data kept at a single source. Each repository requests a subset of data items, with a particular data item chosen with 50% probability. A coherency requirement c is associated with each of the chosen data items. We use different mixes of data coherency. Specifically, the c's associated with data in a repository are a mix of stringent tolerances (varying from \$0.01 to 0.099) and less stringent tolerances (varying from \$0.1 to 0.999). $T\%$ of the data items have stringent coherency requirements at each repository (the remaining $(100 - T)\%$, of data items have less stringent coherency requirements).

$coop\_degree$, the degree of cooperation offered by each repository (i.e., the bound on the number of dependents)—was varied from 1 to 100 in our experiments.

Physical Network – topology and delays: The model for the physical network was randomly generated. It consisting of nodes (routers and repositories) and links, with one of the nodes selected as the source. The routing tables of all the nodes are generated using an all-pairs shortest path algorithm (by Floyd and Warshall [6]). For our experiments, we vary the size of the physical network from 700 nodes to 2100 nodes. Unless specified otherwise, we present results primarily for the 700 node scenario (1 source, 100 repositories and 600 routers). In such a network, an update from one repository (or source) to another traverses around 10 hops on average, compared to the 18 hops reported in [9]. Results for other network sizes are briefly discussed in Section 6.3.5.

Our experiments use node-node communication delays derived from a heavy tailed Pareto [17] distribution: $x \rightarrow \frac{1}{x^{\frac{1}{\alpha}}} + x_1$ where α is given by $\frac{\bar{x}}{\bar{x}-1}$, $\bar{x}$ being the mean and x_1 is the minimum delay a link can have. For our experiments, $\bar{x}$ was 15 ms (milli secs) and x_1 was 2 ms. As a result, the average nominal node-node delay in our networks was around 20-30 ms. This is lower than the delays reported based on measurements done on the internet [9]. We also experimented with higher network delays in Section 6.3.2 and show that the gain in the fidelity using cooperative dissemination is even more significant for higher delays.

Unless otherwise specified, computational delay incurred at a repository to disseminate an update to a dependent is taken to be 12.5 ms. This includes the time to perform any checks to examine whether an update needs to be propagated to a dependent and the time to prepare an update for transmission to a dependent. In the presence of complex query processing at repositories, the time taken to perform the checks can be considerable and hence our choice of computational delay. We also measured the effect of other delay values on fidelity.

Simulation Procedure: After generating the physical network topology, we generate the topology of the d^3t using the technique discussed in Section 4 and conforming, as discussed in Section 3, to the repository's maximum degree of cooperation if specified. The simulation of data dissemination, is then done, using the algorithms discussed in Section 5. Specifically, upon each update to the stock price, the source determines whether to forward the update to the first-level repositories in the d^3t; each repository receiving the update then decides which of its dependents to forward the update.

6.2 Metrics

The key metric for our experiments is the fidelity of the data. Recall that fidelity is the degree to which a user's coherency requirements are met and is measured as the total length of time for which the inequality $|R(t)-S(t)| \leq c$ holds (normalized by the total length of the observations). The fidelity of a repository is the mean fidelity over all data items stored at that repository, while the overall fidelity of the system is the mean fidelity of all repositories.

Rather than computing fidelity, our results plot a more meaningful metric, namely *loss in fidelity*. The loss in fidelity is simply $(100\% - \text{fidelity})$. Clearly, the lower this value, the better the overall performance of a dissemination algorithm.

In addition to fidelity, we also measure the number of updates (messages) sent by each dissemination technique. Clearly, the smaller the number of messages to maintain a certain fidelity, the lower the cost of the coherency maintenance.

6.3 Experimental Results

6.3.1 Baseline Results

Our first experiment examines the efficacy of the d^3t construction algorithm LeLA. We used the source-based algorithm as the baseline data dissemination algorithm.

We consider seven different T values. For each d^3t construction algorithm and these coherency tolerances, we vary the $coop\_degree$ from 1 to 100 and measure the efficacy of the resulting d^3t in providing good fidelity. Note that in the presence of the non-zero communication delays, the structure of the d^3t has a significant impact on fidelity (since the data at a repository is out-of-sync until an update propagates through the d^3t and reaches the repository), the larger the end-to-end delay, the greater the loss in fidelity. As expected, the resulting repository layout network had a maximum diameter of 101 when repositories formed a chain (degree of cooperation=1) and a minimum diameter of 2 when the source updated the repositories directly (degree of cooperation = 100). The average depth ranged from 51 to 1 The average number of dependents varied from 1 to 100.

Figure 3 shows (seen previously in Section 3) that there is a significant loss of fidelity at low values for the degree of cooperation. The loss of fidelity occurs because the d^3t has a large diameter (i.e., a large number of hops between the source and the farthest repository), which increases the communication delays and decreases fidelity.

As the number of dependents of a repository (i.e., the degree of cooperation) is increased, the loss in fidelity decreases to a minima and then starts increasing again. This is consistent with our expectations, since, as explained in Section 3, communication delays dominate when there are a small number of dependents and computational delays at a repository dominate when there are a large number of dependents. The minima occurs when the sum of the two delays is minimized.

The point at which the minima occurs varies slightly from one T value to another and lies between 3 and 20 dependents per repository. (It is worth pointing out that for the communication and computational delay used in this experiments, the value of $coop\_degree$ is 6.

The performance of the algorithm worsens when the number of dependents allowed per repository is increased beyond the optimal value. This is because when the number of permitted dependents is large, the source serves most repositories directly and the d^3t effectively reduces to a one-level tree with most repositories acting as a direct dependent of the source. We explore this behavior further in Section 6.3.2.

Note also that in Figure 3, as the fraction of data items with stringent coherency tolerances decreases, the gradient of the loss in fidelity also decreases.

These results clearly show that, as long as there is some data with stringent coherency requirements, it is important for repositories to cooperate with one another to improve fidelity. Moreover, it is inappropriate to use a very large number of resources towards cooperation. (We elaborate on this point in Section 6.3.3.) Hence, we address the issue of setting the "optimal" level of cooperation in the next section.

6.3.2 Effect of Cooperation on Fidelity

In this section, we thoroughly evaluate the effect of cooperation on fidelity. We begin by showing that if the source is entrusted with the task of disseminating updates directly to repositories, then there is a loss in fidelity, regardless of other system parameters. Thus, it is essential for the source to use the repositories to offload some of its dissemination overheads. We then examine the impact of three key parameters, namely the communication delay, the computational delay and the factor C (see Eq. (2)), on fidelity and demonstrate that when the number of dependents is adapted to communication and computational delays, additional performance benefits can be harnessed.

Performance in the Absence of Cooperation

In the previous section we have already shown that a scenario where the source directly disseminates updates to repositories (i.e., no cooperation between repositories) results in a large loss in fidelity. In this section we show that this result holds regardless of other system parameters. To demonstrate this, we vary the communication delays and the computational delays and assume that the source directly services all repositories in the d^3t. We measure the fidelity offered by he source for different T values. Figures 6.3.2 and 6.3.2 depict our results.

Notice from Figure 6.3.2 that even when we increase the communication delays, fidelity does not drop significantly. This is because, when the source disseminates directly to its dependents, the computation related delays at the source accumulate and the resulting loss of fidelity is primarily due to this effect. When we performed these experiments with only five data items, we noticed, as expected, that the effect of communication overheads was much more apparent then due to the relatively smaller computational delays encountered at the source. In summary, when the number of data items to be handled is large computational delays at the source will have an adverse affect on the scalability of the source. This effect will be pronounced when all repositories desire their data at high coherency (as indicated by the T=100% graph).

This point is corroborated further by Figure 6.3.2. where the loss in fidelity worsens with increasing computational delays, especially when coherency tolerances are stringent.

Controlled Cooperation

When we repeat the scenario whose results were depicted in 3, but with the degree of cooperation chosen as per Eq. (2)), that is irrespective of how many cooperative resources a node has, if offers only $coop\_degree$ resources for its dependents. The performance is as shown in Figure 6.3.2. The behavior becomes an L shaped curve, that is, after the chosen value of $coop\_degree$, loss of fidelity stabilizes.

With controlled cooperation in effect, we studied the impact of communication and computational delays on fidelity. The results (see Figures 6.3.2 and 6.3.2, note that the Y axis goes only from 0 to 10) show that we can counter the effect of large delays in the system by adjusting the degree of cooperation as per the Eq. (2).

In general, these results also show that, using our approach, high fidelity can be obtained even if repository incurs large computation cost (example, if we extend our approach to execute general continuous queries [5]) or when data sizes are large, in which case the communication delays will be larger.

For example, with increasing computational delays, a smaller value of $coop\_degree$ is used (see Eq. (2)), and this reduces the load at a repository; on the other hand, a small value of computational delay results in a larger value of $coop\_degree$. Similarly, with increasing communication delays, a larger value of $coop\_degree$ is used and this will reduce the load on the network; on the other hand, a small value of communication delay will result in a small value of $coop\_degree$. Our results indicate that the degree of cooperation should be higher when the communication delays are large and lower when the computational delays are large.

This clearly demonstrates the benefits of choosing the degree of cooperation based on system overheads for providing high fidelity.

In fact with controlled cooperation performance is not affected by changes to the formulae used to compute the preference factor in d^3t construction algorithm (see Section 4 . Nor is it affected by the exact value of C (see Section 3) used to determine the $coop\_degree$. The results of the experiments are shown in Appendix-E.

6.3.3 Improvement in Fidelity When Coherency Requirements are used to Filter Updates

We have claimed that only updates of interest should be disseminated by a repository to its dependent. In this section, we demonstrate that this filtering is, in fact, essential to achieving high fidelity. To demonstrate this aspect, we compare our approach to a system where *all* updates to a data item are disseminated to repositories interested in that data item. Such a system is emulated by simply using a very stringent coherency tolerance, causing all updates to be disseminated. We compare this system to one where the coherency requirements are not stringent. Less stringent c's result in filtering and selective forwarding of updates. Thus, any difference in performance between these systems is indicative of the fidelity improvement resulting from the filtering that occurs when repositories disseminate only data of interest to their repositories.

Figure 6.3.2 depicts our results. The figure shows that compared to the fidelity of our approach (indicated by the flat "filtered" curve) the approach that disseminate all updates, in fact, results in worse fidelity across the complete range of $coop\_degree$ values. This is because the approach disseminate more messages, which increases the network overheads as well as computational delays at repositories, causing a loss in fidelity. In contrast, intelligent filtering and selective dissemination of updates based on data's coherency requirements can reduce overheads and improve fidelity.

Study of Sensitivity to Parameters of the Tree Construction Algorithm

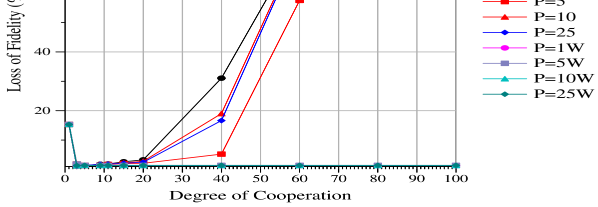
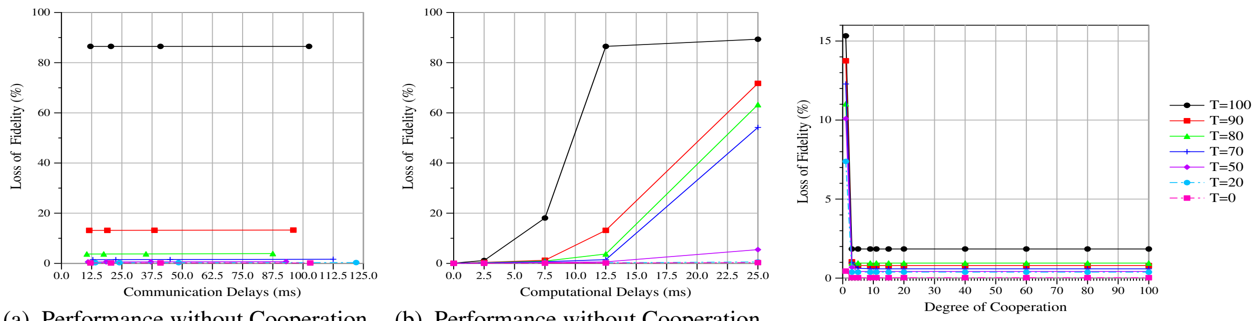


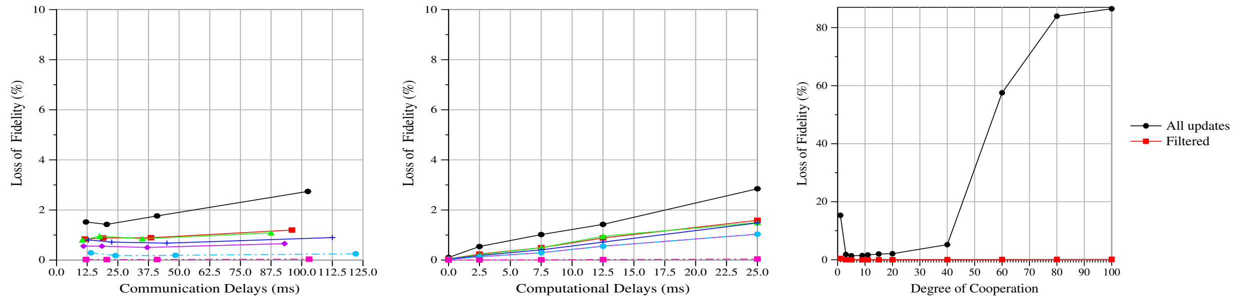
Figure 5: Effect of Different P% Values



(a) Performance without Cooperation, varying Communication Delays

(b) Performance without Cooperation, varying Computation Delays

(c) Performance with Cooperation - Base Case



(a) Performance with Cooperation, varying Communication Delays

(b) Performance with Cooperation, varying Computation Delays

(c) Importance of Filtering during Update Propagation

At each level, the load controller chooses repositories whose preference factor is within $P = 5\%$ of the preference factor for the most preferred parent. We experimented with different values of P (see Figure 5). For $P = 1\%$, the loss in fidelity is high. This is due to the fact that very few of the parents (typically 1) will be serving all the requirements of the dependents. This adds on to the load at the parent and hence fidelity of the system is affected. If a node has a large number of parents (as will be the case for $P > 15\%$, more push connections are used from a certain level (a parent uses one push connection per child, irrespective of the number of data items served to the child) for a single child. Because of this a level can only serve fewer children and this will in turn increase the diameter of the d^3t, again resulting in loss of fidelity. Once the degree of cooperation is chosen, the value of P has little impact on fidelity. This is shown by the curves marked by $P = 1W$, $P = 5W$, $P = 15W$, $P = 25W$.

Our next experiment was to determine if the formula used for calculating the Preference Factor had an impact on fidelity. So we modified the one used so far (See Section 4) to experiment with an alternative $preference\ factor = delay(P, Q) \times numDependents(P)$. This does not account for data availability at a parent. As Figure 6 shows, the choice of the preference function ($P1$ vs $P2$) has insignificant impact on resulting fidelity when the degree of cooperation small.

The results also indicate that once the degree of cooperation is chosen as discussed in Section 3, the parameters of LeLA have little, if any, impact on fidelity obtained. As shown by curves marked by $P1W$ and $P2W$ that for a range of the degree of cooperation values the variation of fidelity is less than 1%. This range depends on the communication and computational overheads. As long as we choose the degree of cooperation from this range, other parameters become secondary for achieving high fidelity.

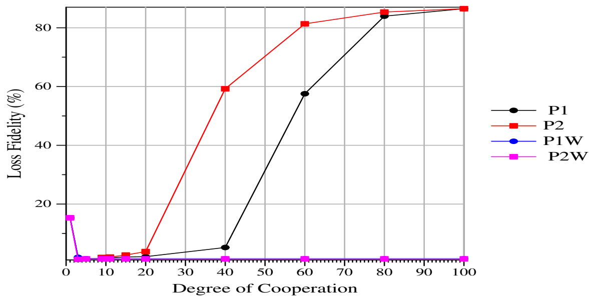


Figure 6: Effect of Different Preference Functions

6.3.4 Performance of Update Dissemination Algorithms

In this section, we compare the performance of the source-based and client-based dissemination algorithms. Figure 7 that the source does nearly 50% more checks of incoming data values to determine if the data value needs to be disseminated to its dependents. As shown in Figure 7, both approaches send the same number of messages through the system and as discussed in Section 5, both approaches guarantee 100%fidelity. So the distributed

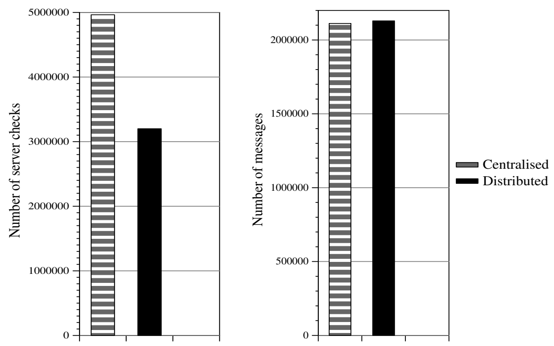


Figure 7: Comparing Centralized and Distributed Data Dissemination Approaches

approach is preferable.

6.3.5 Scalability of the algorithms

We have also studied the effect of increasing the number of repositories on fidelity. Whereas with unlimited cooperation, the diameter of the d^3t could grow to be very high with increasing number of nodes in the network, controlled cooperation limits this growth. For example, when the number of repositories grows from 100 (for the base case) to 300 (and with that the total number of nodes in the system grows from 700 to 2100 nodes), the increase in the loss in fidelity with controlled cooperation was observed to be between less than 5%. This is indicative of the scalability of our set of solutions.

6.4 Summary of Experimental Results

Our performance indicates that

- Each repository should disseminate only updates of interest to its dependents.
- Cooperation is essential to achieve high fidelity and high scalability.
- Cooperation beyond a certain point leads to loss of fidelity. This is because if a repository agrees to disseminate data to too many dependents, queue-ing and dissemination delays at that repository can reduce the fidelity achieved.
- When communication delays and low computational delays are not negligible, the degree of cooperation should be chosen taking communication and computational delays into account because outside the optimal value, the algorithms could lead to increased loss of fidelity.

7 Related Work

Recently several efforts have focused on maintaining consistency between sources and cached copies or replicas. An early work focused on a push-based approach based on expiration times [2]. Achieving transactional consistency among replicas in traditional databases has been studied in [11]. Other efforts that employ push-based techniques include broadcast disks [1] publish/subscribe applications [16], and speculative dissemination [3]. However, the notion of coherency defined in this paper requires a different architecture and algorithms than those in the above efforts.

The problem of selecting an optimal number of replicas has been studied in [18]. Using client-observed round-trip delays as the metric, they show that the payoffs of increasing the number of replicas beyond a certain point are not significant. We focus on a different problem—data dissemination—and use a different metric—data fidelity—to show a somewhat similar result: increasing the degree of cooperation beyond a point is detrimental to fidelity.

Consistency maintenance has also been studied in the context of web caching [14]. In this context, hierarchical web proxy architectures [4] and cooperative web caching [23, 22, 24] have also been studied. The difference between these efforts and our work is that we focus on rapidly-changing dynamic web data while they focus on web data that changes at slower time-scales (e.g., tens of minutes or hours)—an important difference that results in very different solutions. Efforts that focus on *dynamic* web content include [13] where push-based invalidation and dependence graphs are employed to determine where to push invalidates and when. Achieving scalability by adjusting the coherency requirements of data items is studied in [12]. The difference between these approaches and ours is that repositories don't cooperate with one another to maintain coherency.

Mechanisms for disseminating fast changing documents using multicast-based push has been studied in [19]. The difference though is that recipients receive *all* updates to an object (thereby providing strong consistency), whereas our focus is on disseminating only those updates that are necessary to meet user-specified coherency tolerances. Multicast tree construction algorithms in the context of application-level multicast have been studied in [10]. Whereas these algorithms are generic, the d^3t in our case, which is akin to an application-level multicast tree, is specifically optimized for the problem at hand, namely maintaining coherency of dynamic data.

Finally, our work can be seen as providing support for executing continuous queries over dynamically changing data [15, 5]. Continuous queries in the Conquer system [15] are tailored for heterogeneous data, rather than for real time data, and uses a disk-based database as its backend. NiagraCQ [5] focuses on efficient evaluation of queries as opposed to coherent data dissemination to repositories (which in turn can execute the continuous queries resulting in better scalability).

8 Conclusions

In this paper, we examined the design of a data dissemination architecture of repositories that cooperate with one another to maintain coherency of the time-varying data stored in them. The key contributions of our work are:

- Design of a push-based dissemination architecture for time-varying data. One of the attractions of our approach is that it does not require all updates to a data item to be disseminated to all repositories, since each repository's coherency needs are explicitly taken into account by the dissemination algorithm. This intelligent filtering and selective dissemination of updates based on user's coherency tolerances reduces the system-wide network overhead as well as the load on repositories. These in turn improve the fidelity of data stored at repositories.

- Design of mechanisms for maintaining coherency of data within an overlay network of repositories. Our mechanisms were designed to take into account communication delays, computational overheads, and the system load. We also studied their relative performance and showed that cooperation among repositories must be used to improve fidelity substantially with lower overheads, but beyond a certain point, such cooperation can be detrimental to performance.

Whereas our approach uses push-based dissemination, other dissemination mechanisms such as pull [21], adaptive combinations of push and pull [7], as well as leases [8] could be used to disseminate data through our repository overlay network. The use of such alternative dissemination mechanisms as well as the evaluation of our mechanisms in a real network setting is the subject of future research.

References

[1] S. Acharya, M. J. Franklin, and S. B. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of the ACM SIGMOD Conference*, May 1997.

[2] R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Trans. Database Systems*, September 1990.

[3] A. Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time in distributed information systems. In *International Conference on Data Engineering*, March 1996.

[4] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worell. A hierarchical internet object cache. In *Proceedings of 1996 USENIX Technical Conference*, January 1996.

[5] J. Chen, D. Dewitt, F. Tian, and Y. Wang. Niagracq: A scalable continuous query system for internet databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, May 16-18 2000.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. Mc-Graw Hill, 1990.

[7] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramaritham, and P. Shenoy. Adaptive push pull of dynamic web data: Better resiliency, scalability and coherency. In *Procs of WWW 10*, May 2001.

[8] V. Duvvuri, P. Shenoy, and R. Tiwari. Adaptive leases: A strong consistency for the world wide web. In *Proceedings of IEEE INFOCOM 2000*, March 2000.

[9] A. Fei, G. Pei, R. Liu, and L. Zhang. Measurements on delay and hop-count of the internet, 1998.

[10] P. Francis. Yallcast: Extending the internet multicast architecture. http://www.yallcast.com, September 1999.

[11] JoAnne Holliday, Divyakant Agrawal, and Amr El Abbadi. Database replication using epidemic communication. In *6th International Euro-Par Conference*, September 2000.

[12] H.Yu and A.Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *Proceedings of OSDI*, October 2000.

[13] Arun Iyengar and Jim Challenger. Improving web server performance by caching dynamic data. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[14] C. Liu and P. Cao. Maintaining strong cache consistency in the world wide web. In *Proceedings of ICDCS*, May 1997.

[15] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Trans. on Knowledge and Data Engg.*, July/August 1999.

[16] G. R. Malan, F. Jahanian, and S. Subramanian. Salamander: A push based distribution substrate for internet applications. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[17] Mohammad S. Raunak, Prashant J. Shenoy, Pawan Goyal, and Krithi Ramamritham. Implications of proxy caching for provisioning networks and servers. In *In Proceedings of ACM SiGMETRICS conference*, pages 66–77, 2000.

[18] Eric Cronin Sugih Jamin Cheng Jin Danny Raz and Yuval Shavitt. Constrained mirror placement on the internet. *Journal on Selected Areas of Communication (to appear)*, 2002.

[19] Pablo Rodriguez, Keith W. Ross, and Ernst W. Biersack. Improving the WWW: caching or multicast? *Computer Networks and ISDN Systems*, 1998.

[20] S Shah, K. Ramamritam, and P. Shenoy. Maintaining coherency of dynamic data in cooperating repositories. Technical Report LAIIR-DynamicData-0001, IIT Bombay, June 2002.

[21] R. Srinivasan, C. Liang, and K. Ramamritham. Maintaining temporal coherency of virtual warehouses. In *Proceedings of of the 19th IEEE Real-Time Systems Symposium*, December 1998.

[22] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Beyond hierarchies: Design considerations for distributed caching on the internet. Technical Report CS98-04, Department of Computer Science, UT Austin Austin, Texas, USA, 1998.

[23] J. Yin, L. Alvisi, M. Dahlin, C. Lin, and A. Iyengar. Engineering server driven consistency for large scale dynamic web services. *Proceedings of the WWW10*, 2001.

[24] Jian Yin, Lorenzo Alvisi, Mike Dahlin, and Calvin Lin. Hierarchical cache consistency in a WAN. In *USENIX Symposium on Internet Technologies and Systems*, 1999.