Construction of a Coherency Preserving Dynamic Data Dissemination Network

Shweta Agrawal
shweta@it.iitb.ac.in

Krithi Ramamritham
krithi@cse.iitb.ac.in

Shetal Shah
shetals@cse.iitb.ac.in

Indian Institute of Technology, Bombay.

Abstract

In this paper, we discuss various techniques for the efficient organization of a coherency preserving dynamic data dissemination network. The network consists of sources of dynamically changing data, repositories to serve this data, and clients. Given the coherency properties of the data available at various repositories, we suggest methods to intelligently choose a repository to serve a new client request. The goal is to support as many clients as possible, from the given network. Secondly, we propose strategies to decide what data should reside on the repositories, given the data coherency needs of the clients.

We model the problem of selection of repositories for serving each of the clients as a linear optimization problem, and derive its objective function and constraints. In view of the complexity and infeasibility of using this solution in practical scenarios, we also suggest a heuristic solution. Experimental evaluation, using real world data, demonstrates that the fidelity achieved by clients using the heuristic algorithm is close to that achieved using linear optimization. To improve the fidelity further through better load sharing between repositories, we propose an adaptive algorithm to adjust the resource provisions of repositories according to their recent response times.

It is often advantageous to reorganize the data at the repositories according to the needs of clients. To this end, we propose two strategies based on reducing the communication and computational overheads. We evaluate and compare the two strategies, analytically, using the expected response time for an update at repositories, and by simulation, using the loss of fidelity at clients, as our performance measure. The results suggest that a considerable improvement in fidelity can be achieved by judicious reorganization.

1. Introduction

The Internet has grown in popularity from being a mere facility to a necessity. This raises the need for an efficient and scalable dissemination of Internet data to the clients all over the globe. The problem becomes even more challenging when the data is dynamically changing and is used for on line decision making. Examples of such time critical data are many - stock prices on finance sites, weather information, sports data, sensor data, etc. We focus on ways and means to distribute such dynamically changing (streaming) data to a large number of users with high accuracy, efficiency, and scalability.

A natural solution of the problem of efficient distribution is to introduce some repositories that replicate the data between the sources and clients. These repositories can serve the clients geographically closer to them and reduce the load on sources. But, for rapidly changing data, the overheads can be very large for achieving replication.

Fortunately, we can and should exploit the fact that different users have different requirements for the accuracy of data. The user can specify the bound on the tolerable imprecision for each requested data-item; this can be viewed as the *coherency requirement* associated with the data. For example, a stock broker might be concerned with every cent of change in stock prices while a casual user might be content with a much lower accuracy. The goal is to provide a user with data at the desired accuracy. The fraction of total time for which the client receives the data at the desired precision is called *fidelity*. (Formal definitions of *coherency requirement* and *fidelity* are given in Section 2.1.)

Meeting user coherency requirements when the data is changing rapidly and unpredictably is a challenging problem. We had previously developed an algorithm to construct a cooperative repository network for dynamic data which is coherency-preserving, resilient to failures, and scalable to a large number of data-items and clients [11]. The focus of this algorithm, called DiTA, is on maintaining coherency of dynamic data-items in a network of repositories: data disseminated (pushed) to one repository is filtered by that repository and disseminated to the repositories dependent on it, according to their respective data and coherency requirements. Section 2.2 gives a brief description of DiTA.

This paper makes two key contributions.

1. Given a repository network, we solve the problem of determining the repository to which a client should connect in order to satisfy its data needs.
2. Given the data and coherency needs of clients in the network, we propose techniques to reorga-

nize the repository network, so that the clients receive the data at a better fidelity.

Our algorithms assume accurate, global knowledge about the load on repositories and distance of clients to repositories, at the source. The design and evaluation of a scalable, distributed architecture for implementing these algorithms is a topic for future research.

1.1. Assigning clients to repositories

In Section 3.1, we formulate the problem of assigning client requests to repositories as a linear optimization problem. Given the complexity of the LP solution (exponential in the worst case), we also present a heuristic solution in Section 3.2. In this solution, the source of a data-item maintains the information about the availability and coherency values of the data-item on all the repositories in the network, using which it chooses the appropriate repository to serve a request. Experimental evaluation using real world traces, in Section 3.3, demonstrates that the fidelity achieved by clients, using the heuristic assignment, is close to that achieved using linear optimization.

Effective load sharing among repositories can improve the fidelity experienced at clients by decreasing the average computational delay at the repositories. The *resource contribution limit*, which denotes the maximum number of requests a repository can serve, helps controlling the load at the repositories. In Section 3.4, we propose an algorithm for adaptive adjustment of individual resource contribution limits of repositories during assignment of client requests, for better load sharing.

1.2. Reorganizing the repository network

In a practical scenario, client needs change continuously: the data-items and the coherency requirements of the clients can change. The repositories in the network should be able to adapt to these changes and serve the data as per the current needs of the clients. In Section 4, we present and evaluate two strategies to reorganize the data served by the repositories according to the current clients' requirements.

- The first strategy, called the *Closest Repository* algorithm, is based on reducing the fidelity loss due to network communication delay, by making the data required by a client available at a repository close to it.
- The second strategy, called the *Divide and Conquer* strategy, extends the *Closest Repository* approach by categorizing requests into multiple classes, according to their respective coherency requirements. A fraction of repositories is reserved for serving each class of requests. This strategy further improves fidelity by reducing the average computational delay at repositories.

We evaluate and compare the two strategies, analytically, by calculating the expected response time for updates at repositories, and by monitoring the loss of fidelity at clients during simulation. We observe that the fidelity achieved using the *Divide and Conquer* method depends on the relative size of the fraction of repositories reserved for serving each class of requests. We show that using the fraction size at which the analytically calculated response time is minimum, the *Divide and Conquer* strategy gives a significant improvement in fidelity over the *Closest Repository* approach.

In Section 5, we present the overall algorithm for the construction of dynamic data dissemination network, making use of the above algorithms and strategies.

In summary, this paper presents strategies for efficient organization of a coherency preserving dynamic data dissemination network. The word "dynamic" here applies to "data" as well as "network". We construct a network for the distribution of *dynamically* changing data to the clients according to their coherency requirements. The network also periodically changes its organization according to the requirements of the clients.

We give an overview of the related work in Section 6, and present the conclusion and future work in Section 7.

2. Background: The Basic Framework and DiTA algorithm

2.1. Data coherency and overlay network

As shown in Figure 1, we build a push based data dissemination network of sources and repositories with clients connecting to the repositories. To maintain coherency of the

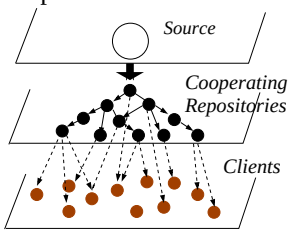


Figure 1: The network architecture

cached data at repositories, each data-item must be periodically refreshed with the copy at the source. Let the client specify a *coherency requirement*(cr) for each data-item of interest. The value of cr denotes the maximum permissible deviation of the value of the data-item at the client from the value at the server, and thus constitutes the user-specified tolerance. Observe that cr can be specified in units of *time*, e.g., the data-item should never be out-of-sync by more than 5 minutes ("time domain consistency requirement", similar to the data validity interval [9]) or *value*, e.g., the temperature should never be out-of-sync by more than one degree ("value domain consistency requirement"). In this paper, we only consider coherency requirements specified in terms of value of the object; maintaining coherency requirements in units of time is a simpler problem that requires less sophisticated techniques (e.g., push every 5 minutes).

Formally, let $S_d(t)$ and $C_d(t)$ denote the value of data-item d at the source and the client, respectively, at time t. Let c_d be the coherency requirement of the client for d. Then, to maintain coherence, we should have

$$\forall t,\ |C_d(t) - S_d(t)| \leq c_d$$

Empirically, *fidelity* f observed by a client can be defined as the total time for which the above inequality holds, normalized by the total length of observations. The goal of our dissemination network is to provide high fidelity at low communication and computational overheads.

For each data-item, we build a logical overlay network, as follows. Consider a data-item d served by a source S. The source directly serves some of the repositories. These repositories in turn serve a subset of remaining repositories and a subset of clients, such that the resulting network is in the form of a tree rooted at the source, and consisting of repositories and clients interested in d. This tree is referred as the *dynamic data dissemination tree*, or d^3t for d. The children of a node in the tree are also called the *dependents* of the node. When a data change occurs at the source, it checks which of its direct or indirect dependents are interested in the change, and pushes the change to them. Each repository acts as a filter and sends only the updates of interest further down.

2.2. DiTA: Data-item-at-a-Time Algorithm

Here is a short description of the DiTA algorithm [11] to insert a repository with given data-items and coherency requirements, in the overlay network. A repository P interested in data-item d with coherency c requests the source of d for insertion. If the source has resources to service P, it is made a child of the source in the d^3t for d. Otherwise, the source determines the best subtree rooted at its children for the insertion of P. The subtree is chosen such that the level of P in the d^3t is the smallest possible and the communication delay between P and its parent is small. This is recursively applied to select subtrees in this subtree, till a repository Q is chosen that has resources to serve P. If the coherency requirement of Q is less stringent than P, P pushes it down in the subtree and replaces Q. This ensures that the repositories with more stringent coherency serve repositories with less stringent coherency in the d^3t for d. DiTA requires very little book-keeping and experimental results in [11] show that it indeed produces d^3t of repositories that delivers data with high fidelity.

3. Assigning clients to repositories in a given network

In this section, we focus on the problem of assigning new client requests to an existing dynamic data dissemination network. If all the client requests were known in advance, the task of assignment could be seen as solving an optimization problem, as described in Section 3.1, where the average fidelity observed by clients has to be maximized, subject to the constraints of data-availability and resource limitations at the repositories. In practice, however, the client requests are expected to arrive one at a time, and have to be assigned to some repository as soon as they come. Moreover, the complexity of solving such an optimization problem also makes it inappropriate for use as and when the requests arrive. Still, this solution can be used to estimate the goodness of the heuristic solution which we describe in Section 3.2, that handles one request at a time.

3.1. Assignment using linear optimization

Assumptions:

1. Each repository has a fixed resource contribution limit.
2. Repository to client communication delay is negligible. [1].
3. The basic capacity of the repositories (computational delay for processing a single update) is same.

Problem definition: Given a set of repositories, the data-items served by each of them, and a set of clients with their respective data-item requirements, the aim is to find a three dimensional 0/1 matrix x that gives a mapping between the repositories and the <client, data-item> requests. A "1" value at a position (p, c, d) of x recommends that the request for the data-item d by client c should be assigned to the repository p.

Input information about the dissemination tree: The following information about the d^3t of repositories and about the clients to be inserted is provided as input to the optimization problem.

- *served:* Two dimensional matrix giving the coherency values of data-items available at the repositories. A positive value of $served(p, d)$ denotes the coherency value at which data-item d is available at repository p. A negative value indicates that p does not serve d.
- *request:* Two dimensional matrix giving the coherency requirements of the data-items requested by the clients. A positive value of $request(c, d)$ denotes the coherency with which client c needs data-item d. A negative value indicates that client c has not requested d.
- *max_resources:* Single dimensional matrix giving the resource contribution limit of each repository, determined by the number of data-items available at the repository, multiplied by a parameter R. The resource contribution limit helps in spreading the load across multiple repositories by limiting the number of requests a repository can serve.

1 This assumption applies only for the optimization problem, in order to simplify the objective function.

Constraints: The optimization constraints are designed to ensure the data-availability, resource limit and load balancing at the repositories.

1. If a data-item d is not available at a repository p, or a client c does not request data-item d, the value of $x(p, c, d)$ is set to zero.
2. Eliminating redundancy in serving requests: Not more than one repository should serve a single data-item request from a client.
$$\forall c, d: \qquad \sum_{p} x(p, c, d) \leq 1$$
3. Resource contribution limit: The number of client requests i.e., <client, data-item> pairs served by a repository, must be less than or equal to the resource contribution limit of the repository.
$$\forall p: \qquad \sum_{c,d} x(p, c, d) \leq max\_resources(p)$$
4. Serving maximum possible requests: The total number of requests served for a data-item d must be equal to total number of requests made for d ($total\_requests_d$), or to total resource contribution limit for d over all repositories ($total\_res_d$), whichever is smaller. This ensures that the system serves as many requests as possible.
$$\forall d: \sum_{p,c} x(p, c, d) = min\{total\_requests_d, total\_res_d\}$$
5. Load balancing: If too many requests are assigned to a repository, the computational delay at the repository (to check and push the updates) will increase which will result in loss of fidelity of the dependent clients. The number of requests should be balanced among repositories. But, the resource contribution limits of repositories can be different in which case they may not serve an equal number of requests, hence, some deviation must be allowed. We try to achieve load balancing by introducing the following constraint:
$$\forall p: \qquad (1 - D) \leq \frac{\sum_{c,d} x(p, c, d)}{average\_requests} \leq (1 + D)$$
where, $average\_requests$ is given by
$$\frac{min\,(total\,requests,\ total\,resource\,limit)}{number\,of\,repositories}$$
and D is the deviation from the average. The value of D depends on the deviation of $max\_resources$ of various repositories from the average value. The more the deviation, the greater should be the value of D to prevent any conflict with constraint 3. Empirically, the value of D is kept as 0.5.

Objective function: The objective function should reflect the need to maximize the fidelity achieved by the clients. But without the knowledge of the actual changes in values of various data-items, we cannot calculate the actual fidelity achieved. Ideally, maximum data-item requests should be served at the required coherency values, which implies that the ratio of requested to served coherency value for a data-item should be as close to "1" as possible. An objective function with the above target is as follows:
$$maximize: \sum_{p,c,d} RS\_ratio \times x(p, c, d)$$
where
$$RS\_ratio = min\left(\frac{request(c, d)}{served(p, d)}, 1\right)$$
For example, a client request for 0.05$ coherency can be served by 0.05$ or 0.01$ coherency data, either of the choices leads to an addition of "1" to the objective function. There is no extra gain in the objective function by serving with more stringent coherency(0.01) than required(0.05). This objective ensures that the served coherency is as close as possible to the requested coherency, but not less.

3.2. Assignment using heuristics

The heuristic algorithm handles the dynamic arrival of client requests, inserting one request at a time in the network. In this approach, there is a *selector* node for each data-item d. The *selector* maintains a list of repositories serving d, sorted in the order of coherency values at which d is available at these repositories. In the DiTA algorithm [11], the source already stores such information to maintain the coherency of data-items at the repositories. So, the source of a data-item can act as a *selector* for it. Each request initially contacts the *selector*, which then directs it to the appropriate repository.

When a request from a client C for a data-item d at coherency value c arrives at the *selector*, it goes through the list of repositories serving d and having resources to serve a new request (they have not exceeded their resource contribution limits), and short-lists those serving d at a coherency close to c. Among those shortlisted, a repository P is selected such that the sum of the computational delay (estimated by the number of client dependents) at P and the communication delay between the client C and repository P is the smallest possible.

Experimental results, discussed in the next section, indicate that the fidelity achieved by assignment of client requests using this simple approach is close to that achieved using linear optimization.

We now discuss the space and computational overheads of this approach. This algorithm involves storing and maintaining the sorted list of coherencies at the *selector*, and finding the repositories with coherency values close to c. In case of a large number of repositories, the overheads at the source can be prohibitive. To avoid this scalability problem, the computation can be moved to the repositories served directly by the source. Each such repository will be at the first level of the d^3t and act as *selector* for a subset of data-items. The source updates these selectors when there are changes

in the coherency values of data available at the repositories in the network. When a new <client, data-item> request arrives, the source passes it to the repository acting as the *selector* for that data-item.

3.3. Performance of the assignment algorithms

3.3.1. Experimental methodology The performance of our solutions is investigated using real world stock price streams as exemplars of dynamic data. The presented results are based on stock price traces obtained by continuously polling *http://finance.yahoo.com*. We collected values for 1000 stocks making sure that the stocks did see some trading during that day. [11] gives the details of some of the traces used. For each of the traces, a new data value was obtained approximately once per second. Since stock prices change at a slower rate, the traces can be considered to be "real-time" traces.

We simulated the situation where all the repositories and clients access data kept at one or more sources. Each of them 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. The c's are a mix of stringent and lenient tolerances. A request for a data-item has a coherency requirement chosen uniformly from the stringent range with probability T and from the lenient range with probability $(1-T)$. Unless otherwise stated, the stringent range is kept as \$0.01 to \$0.05 and the lenient range as \$0.5 to \$1, and T as 0.5 for our experiments in this paper. The results stated are an average over 5 or more independent runs of the simulation. Each run involves using a different set of traces for the data-items in the simulation.

The physical network consists of nodes (routers, sources, repositories & clients) and links. The underlying router topology was generated using BRITE (http://www.cs.bu.edu/brite). The clients, repositories and the sources were randomly placed in the router plane and connected to the closest router. For each client and repository, a set of data-items of interest was generated and then the coherencies were chosen from the desired range.

Our experiments use node-node communication delays derived from a heavy tailed Pareto [10] 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 1.5 ms and x_1 was 0.2 ms. 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 [4].

The 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.

For solving the linear optimization problem of mapping, the GNU Linear programming kit (GLPK) (*http://www.gnu.org/prep/ftp.html*) was used.

3.3.2. Performance Metrics The key metric for our experiments is the loss in fidelity experienced by the clients: *fidelity* is the degree to which a client's coherency requirements are met. It is measured as the total length of time for which the inequality $|C_d(t) - S_d(t)| \leq c_d$ holds (normalized by the total length of observations), where $C_d(t)$ is the value of data-item d at the client and $S_d(t)$ is the actual value at the source at time t. The fidelity of a client is the mean fidelity over all data-items requested by that client, while the overall fidelity of the system is the mean fidelity of all clients. Loss in fidelity is simply $(100\% - fidelity)$.

3.3.3. Comparison of linear optimization and heuristic algorithm Figure 2 shows the simulation results for various number of clients for a network with 2 sources, 20 repositories and 40 data-items. To get an idea of the scale of the problem, let us calculate the total number of requests, given the number of clients and data-items in the network. Since each client requests a subset of data-items, with each data-item chosen with 50% probability, the average number of data-items requested by a client will be half of the total number of data-items available. Therefore, the total number of requests in the network with c_{total} clients and d_{total} data-items will be approximately $\frac{c_{total} \times d_{total}}{2}$. For example, for 500 clients and 40 data-items, number of requests will be approximately 10,000.

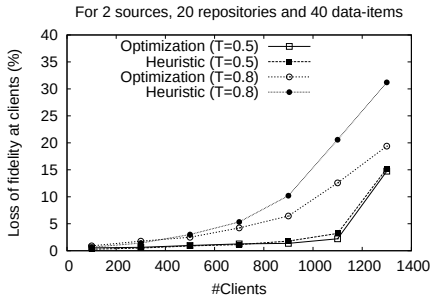


Figure 2: Fidelity loss for client assignment approaches

As seen in the figure, the heuristic assignment algorithm gives a loss of fidelity very close to that achieved by linear optimization at $T = 0.5$. At $T = 0.8$, the coherency requirements of the clients are more stringent. The linear optimization approach, aware of all the requests in advance, is able to keep the stringent resources for the stringent requests, and achieves a better overall fidelity.

Also, Figure 2 shows a sudden rise in the loss of fidelity as the number of clients in the network increases beyond 1000. This is due to the fixed hard limit of resources at each repository. The resource contribution limit, or the maximum number of requests served by a repository, is given by R multiplied by number of data-items available at the repository ($max\_resources$, see Section 3.1). The value of R is

50 for this simulation. As the number of client requests exceeds the total resource contribution limit of repositories, the requests start getting dropped. The fidelity experienced by client is zero for each dropped request. This causes a quick drop in the average fidelity.

3.4. Setting the resource contribution limit of the repositories

Resource contribution limit is a way by which the load on a repository can be controlled. But as seen above, if R, and hence the resource contribution limit, is too small, a large number of requests can get dropped.

Figure 3 shows the effect of the change in value of R on the average fidelity achieved by clients. In this simulation, the clients join and leave the network dynamically. The inter-arrival and inter-departure time of clients follow exponential distribution. The arrival rate λ is 2000 times the departure rate μ and the average number of clients at a time in the system is 2000. (This can be viewed as an $M/M/\infty$ queuing system. The average number of clients for such a system is given by $E[N] = \frac{\lambda}{\mu}$ where N is number of jobs in the queuing system). The curve labeled "Fixed resource contribution limit" indicates high loss in fidelity when R is kept below 100. The loss in fidelity remains at about 12% at all values of R above 100. Thus a small R causes the requests to drop, but a large R does not affect the performance. This is because the resource contribution

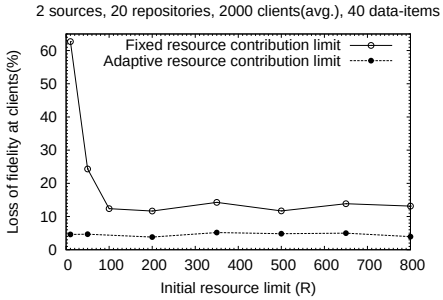


Figure 3: Resource contribution limit of repositories

limit, determined by a common parameter R, only attempts to make the number of requests served by the repositories equal. This is already taken care by considering the number of client dependents on a repository while selecting a repository in the heuristic client assignment algorithm (and by the load-balancing constraint #5 in the linear optimization method).

Note that even when we assume all the repositories to be identical in their basic capacity (computational delay for each update), some repositories may be able to serve less clients than others. For example, those which are near the root of the data dissemination tree may be more loaded. We should be able to determine the individual capacity and hence the resource contribution limit of various repositories on the fly, so that the more capable repositories can share the load of the less capable ones.

3.4.1. Adaptive algorithm to adjust the resource contribution limit The response time for an update gives an indication of the load on a repository. Varying the resource contribution limit of repositories according to their recent response time can help in controlling the load at the repositories. This leads us to the following algorithm for resource contribution limit adjustment:

- *Adaptive increase:* When the resource contribution limits of all repositories serving a data-item are exhausted, the selector sends a message down the dissemination tree to the repositories to increase their limits. A repository with small response time for updates is expected to be less loaded than one with a large response time, and can serve more clients. Thus, on receiving selector's message, the repositories increase their resource contribution limit by amount *inversely proportional* to their recent average response time.
- *Adaptive decrease:* The selector node periodically monitors the difference between the total resource contribution limit of all repositories and total number of client requests in the network. If the total limit is much more than total number of requests, it sends a message to all repositories to decrease their limits. A large response time at a repository indicates heavy load. Thus, on receiving the selector's message for decrease, the repositories decrease their future resource contribution limit by an amount *directly proportional* to their recent average response time. This reduces the future assignment of clients to the overloaded repositories.

  Note that a repository may reduce its future resource contribution limit to a number less than the number of requests already assigned to it. No reassignment of extra requests is done; but as the clients join and leave, in due course, the requests assigned to the repository is reduced to the desired number. Also, if at some point, the total resource contribution limit of all the repositories goes below the total requests in the network, the *selector* node will automatically detect this and call for an "Adaptive increase".

3.4.2. Performance of the adaptive algorithm The adaptive algorithm achieves better performance through better load sharing techniques, as is evident from the curve labeled "Adaptive resource contribution limit" in Figure 3. The variation of the resource contribution limit of repositories was examined over time. It was seen that the resource contribution limit of the source remains at a value lower than that of the repositories. The algorithm detects that the assignment of clients to the source increases its response time considerably, and keeps the resource contribution limit at the source low. This suggests that it might be beneficial not to assign any clients to the sources.

We compared the fidelity obtained [1] in case the requests are assigned only to sources, assigned to sources and repositories both, and only to the repositories. The experiments showed that the loss of fidelity rises rapidly if we have no repositories in the network; the computational delay at the source, to push the updates to a large number of clients, results in high loss of fidelity. Further, it is seen that better fidelity can be achieved if the client requests are assigned only to repositories.

For subsequent experiments, we assign the client requests only to the repositories in the network, using the heuristic assignment algorithm. But, as these experiments are snapshot based (all the repositories and clients are inserted in the network and then the updates are simulated), we do not use the adaptive algorithm for adjusting the resource contribution limit. However, it is kept sufficiently large such that the requests are not dropped.

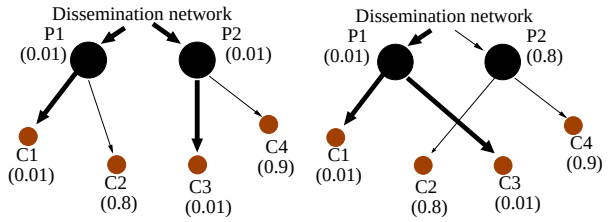
4. Reorganization of a repository Network

Ideally, the choice of data-items available at the repositories should be driven by the data needs of the clients, and the repositories should be able to adapt to the changing demands for the data-items. In this section, we address the following problem: Given the needs of the current clients in the network, determine the data-items to be served, and their respective coherency values, at each repository. The goal is to minimize the loss of fidelity experienced at clients. Fidelity loss is mainly due to two kinds of delays in receiving updates - the communication delay, and the computational delay at the repositories. In the following two strategies, we try to deal with each of these delays.

4.1. Reducing the communication delay: the Closest Repository algorithm

In a widely distributed network, the delay in communication from a repository to a client can affect the fidelity experienced by the client, significantly. The *Closest Repository* algorithm is based on reducing this communication delay by making the data required by a client available at a repository close to it (in terms of network link delay).

In this algorithm, a data-structure is created that maps each client request to a repository close to it. Instead of mapping the request to the closest repository, to balance the number of requests mapped to various repositories, a set of closest repositories is short-listed and the request is mapped to the repository with the least number of already mapped requests. The set of data-items served by a repository is derived as the union of the data-items in the requests mapped to the repository. The repository serves each of these data-items with the most stringent of the coherency values demanded for it in the mapped requests.



(a) Closest Repository (b) Divide and Conquer

Figure 4: Coherency requirements of repositories for the two approaches

4.2. Reducing the computational delay: Divide and Conquer strategy

In case of large number of clients, the requests assigned to a repository are likely to cover a wide range of coherency values. To satisfy these requests, the repository has to obtain the data from its parent at the minimum of these coherency values. As a result, the repositories end up receiving almost all the changes. This defeats the purpose of coherency based dissemination. For example, consider Figure 4(a) where a repository P_1 serves a data-item d at 0.01 coherency to a client C_1, and at 0.8 coherency to C_2. Similarly, P_2 serves d at 0.01 to C_3, and at 0.9 to C_4. Both the repositories have to receive d at a coherency of 0.01 from the source.

Instead of each repository serving all kinds of requests, suppose that the requests are divided into various classes according to their coherency demands, and a different fraction of repositories is reserved for serving each of them. The network can now take a better advantage of the coherency based dissemination. For example, as shown in Figure 4(b), if the requests for 0.8 and 0.9 coherency values are assigned to P_2 and both the 0.01 coherency requests to P_1, P_2 will need to get d at 0.8 coherency only.

This strategy of categorizing the client requests into various classes according to their respective coherency demands, and reserving a fraction of repositories for each of them, is called the *Divide and Conquer* strategy. In this strategy, the data served by a repository is derived using the *Closest Repository* approach applied *within a class*, i.e., the requests are mapped to one of the closest repositories within the request's class only; the data-items served and their respective coherency values are derived from the mapped requests in the same way as before. Similarly, a repository to assign a new client request is selected using the *heuristic client assignment* algorithm (Section 3.2), only from the repositories in the request's class.

Divide and Conquer strategy improves the fidelity by reducing the computational delay at repositories:

- *Less traffic in the repository network:* Coherency-based classification allows repositories serving the class of lenient requests to have the data at less strin-

gent coherency values. This results in less overall data traffic in the network. Decrease in data traffic implies decrease in the number of updates to be processed by the repositories, which leads to lower delays at the repositories, and a better fidelity at clients.

- *Load sharing:* The load on a repository due to a data-item request depends on the coherency value of the data-item. A stringent request is expected to incur more load on a repository than a lenient request since more updates will have to be transmitted for a stringent request. Reserving a larger fraction of repositories to serve stringent requests can help in a better sharing of load among repositories which reduces the average computational delay at repositories.

4.3. Evaluation of the reorganization strategies using analytically calculated response time

For a large number of clients and data-items, the computational delay dominates other delays. The expected response time reflects the loss of fidelity due to computational delay. So, the expected response time can be used to compare the two reorganization strategies.

For the *Divide and Conquer* strategy, we consider the case when the requests are divided into two classes. The *tight* class is defined as the class of requests with coherency demands in the stringent rang, and the *loose* class as requests with lenient coherency demands. A fraction x of the total repositories are reserved for the *tight* class, while the remaining $(1-x)$ fraction of repositories serve the requests in the *loose* class. x is termed as the *cut-off* value of the division. The expected response time for the *Divide and Conquer* strategy depends on the value of *cut-off* x. For complete derivation, and formulae of the expected response times for the two strategies, please refer to [1].

Figure 5 shows a plot of the expected response time for the *Closest Repository* (C R) and *Divide and Conquer* (D & C) strategies, as a function of the *cut-off* x. The stringent range of coherency values is taken as $(0.01, tight_{max})$ and the lenient range as $(0.5, 1)$, with $T = 0.5$. x is varied from 0.5 to 1. The expected response time is plotted for various values of $tight_{max}$ (the values listed in brackets). Some important observations from the plots and the results listed in Table 1, are as follows:

1. The value of the expected response time at repositories, in case of *Divide and Conquer* strategy, is highly sensitive to the value of *cut-off* x. If x is small, the repositories serving the *tight* class of requests are congested with load, and their response time increases; if it is big, the *loose* class of requests do not have enough repositories to handle their load. There is an optimal value of x, at which the average response time is minimum.

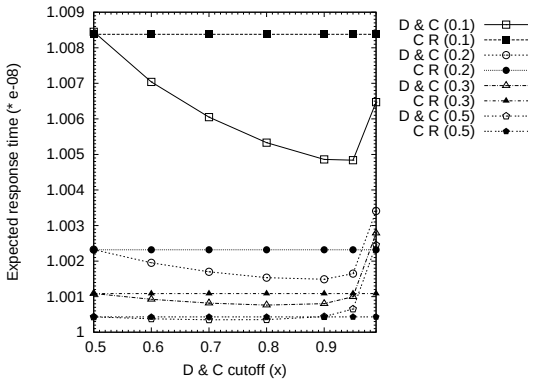


Figure 5: Expected response time as a function of cut-off x (D & C: Divide & Conquer, C R: Closest Repository)

Table 1: Optimal cut-off using analytical model

$tight_{max}$	optimal (x)
0.05	0.95
0.1	0.95
0.2	0.9
0.3	0.8
0.4	0.8
0.5	0.7

(a) $(0.01, tight_{max})$, $(0.5, 1)$, $T = 0.5$

T	optimal (x)
0.2	0.4
0.3	0.6
0.4	0.7
0.5	0.7
0.6	0.8
0.7	0.9
0.8	0.9

(b) $(0.01, 0.5)$, $(0.5, 1)$

2. At the optimal value of x, the expected response time in case of the *Divide and Conquer* strategy is significantly less than the expected response time in case of the *Closest Repository* algorithm.

3. The optimal value of x decreases as we loosen the tight coherency range by increasing $tight_{max}$. Table 1(a) lists the optimal values of x for various values of $tight_{max}$. For larger values of $tight_{max}$, the *tight* class of requests being less stringent, incur less load, and require a smaller fraction of repositories to serve them.

4. Table 1(b) shows the optimal values of x for different values of T with the tight and loose coherency ranges taken as $(0.01, 0.5)$ and $(0.5, 1)$ respectively. As T increases, more requests are expected to have their coherency demands in the stringent range, hence more repositories are needed for the *tight* class, and the optimal value of x shifts towards 1. Note that, even at $T = 0.5$ (uniform coherency distribution), the optimal value of x is greater than 0.5. More repositories are needed to serve stringent requests than are needed for an (expected) equal number of lenient requests.

4.4. Evaluation of the reorganization strategies using simulation

Here, we study the performance of the two reorganization strategies by simulation using real world stock price

streams. The experimental methodology for the simulation is the same as described in Section 3.3.

As in the previous section, the coherency distribution is a mix of coherency values from the stringent range $(0.01, tight_{max})$ and the lenient range $(0.5, 1)$ with probability $T = 0.5$. The requests are divided into *tight* and *loose* classes with a fraction x of the repositories serving the *tight* class of requests and $(1 - x)$ of the repositories reserved for the *loose* class. Figure 6(a) shows the loss of fidelity experienced at the clients during simulation, for the two reorganization strategies, as a function of the *cut-off* value x; where x varies from 0.5 to 1.0. The loss of fidelity is plotted for various values of $tight_{max}$ (values in brackets).

Figure 6(a) shows that the variation in loss of fidelity experienced at clients during simulation is similar to the variation of analytically calculated expected response time shown in Figure 5. The optimal value of x, at which the analytically calculated response time is minimum, is found to be very close to the value of x at which the loss of fidelity achieved by clients is minimum. Thus, the optimal *cut-off* x calculated by minimization of analytically calculated response time can be a good estimate of the optimal value of x in a real system.

Figure 6(b) and 6(c) show the effect of reorganization on loss of fidelity at clients, for various number of clients and data-items in network, respectively. The curve labeled "Without reorganization" shows the case when the data-items served by repositories are randomly determined. Each repository serves a data-item with 50% probability, at a coherency value selected randomly from the stringent range $(0.01 - 0.05)$ with probability T, and from the lenient range $(0.5 - 1.0)$ with probability $(1 - T)$, where $T = 0.5$. The requirements of clients are also generated in the same fashion. The curve labeled "Closest Repository" shows the case when the requirements of clients were generated in the same way as above, but the data available at a repository was derived from the requirements of clients using the *Closest Repository* approach. For the curve labeled "Divide & Conquer", we use the *Divide and Conquer* strategy to divide the requests and repositories into the *tight* and *loose* classes. The *cut-off* x is chosen as the value at which the analytically calculated response time is minimum for the given coherency distribution of client requests.

As shown in Figure 6(b) and 6(c), a considerable improvement in fidelity of clients is achieved by reorganizing the data available at the repositories using the *Closest Repository* algorithm. The *Divide and Conquer* strategy enables the network to take advantage of coherency based dissemination even on larger loads by reducing the average computational delay at repositories. The figure shows that for high number of clients or data-items, the loss of fidelity at clients using the *Divide and Conquer* strategy is considerably less than that using the *Closest Repository* algorithm. For example, in Figure 6(b), for 700 clients, the fidelity achieved by *Divide and Conquer* strategy is about 20% more than that achieved by the *Closest Repository* algorithm and about 40% more than achieved without reorganization. Similarly, in Figure 6(c), for 500 data-items, the improvement is around 10% and 20% respectively.

Thus, by reorganizing the data at repositories, the network can accommodate more clients and data-items. For example, suppose we want to serve the clients with 90% fidelity on average. A network with 1 source, 10 repositories and 100 data-items will be able to serve only 250 clients without any reorganization; about 450 with reorganization using the *Closest Repository* approach; by using *Divide and Conquer* strategy, it can accommodate over 700 clients providing the same fidelity.

5. The overall algorithm

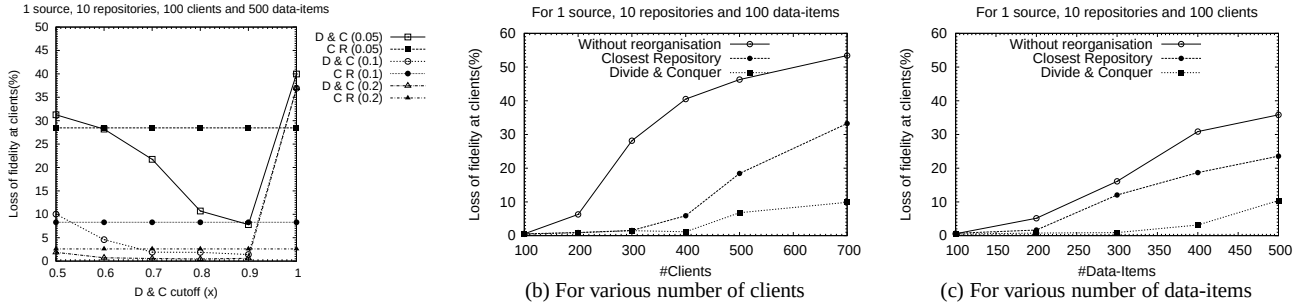
Overall, we suggest the following scheme for the construction of the dynamic data dissemination network:

1. Calculate the *Divide and Conquer cut-off* value x using the analytical model of response time.
2. Reserve x fraction of the total repositories for serving the *tight* class of requests and the remaining $(1 - x)$ fraction for the *loose* class.
3. Determine the data to be served by the repositories in each class from the client requests in that class using the *Closest Repository* approach.
4. Insert the repositories in the network, using the DiTA algorithm [11].
5. Assign clients to the repositories in their class using the *heuristic assignment* algorithm (with adaptive adjustment of resource contribution limits of repositories).
6. Reorganize the network periodically by calculating the new *cut-off* value x, and determining the requirements of the repositories from the current data and coherency requirements of clients.

At present, we divide the requests into two classes only; we intend to study the performance in case of multiple classes, in future. Also, an interesting exercise will be to find an algorithm for the estimation of time period after which the data available at the repositories should be reassigned, and the repositories reorganized.

6. Related Work

Work on scalable and available replicated servers [12], and distributed servers [3] are related to our goals. [12] addresses the issue of adaptively varying the consistency requirement in replicated servers based on network load and application specific requirements. Our work on the building, reorganization and dissemination of dynamic data in a network based on the coherency requirements of the clients.



(a) Fidelity loss as a function of cut-off x (D & C: Divide and Conquer, C R: Closest Repository)

(b) For various number of clients

(c) For various number of data-items

Figure 6: Effect of reorganization on fidelity

The data at a repository is not exactly a replica of the data at the source rather it can be seen as a projection of the sequence of updates seen at the source.

The concept of approximate data at the users was studied in [7, 8]; the approach focuses on pushing individual data-items directly to clients from the source, based on client coherency requirements. We consider the cooperative repository based dissemination network in between the source and clients and show that by intelligently choosing the repository to which a client should connect, a considerable improvement in fidelity can be achieved.

The scheme of prioritizing the requests or jobs on the basis of their time criticality has been proposed in [5], for scheduling soft-real time jobs on dual non-real time servers. In the *Divide and Conquer* strategy, we extend this concept from time constraints to *value* constraints specified as coherency requirements.

7. Conclusion

In this paper, we address various challenges in the organization of a coherency preserving dynamic data dissemination network. The key contributions of this paper are

- Algorithms to intelligently choose a repository for a new client request. We solve the request assignment problem as an optimization problem, and also suggest a more practical heuristic solution that can handle dynamic arrival of requests. We show that the performance achieved by the heuristic assignment is close to that achieved by the linear optimization approach.
- Algorithm to adjust the resource contribution limit of repositories adaptively according to the recent average response time. This helps in better sharing of load among repositories.
- Mechanisms for reorganization of the repository network, to make it adapt to the changes in client requirements. The *Closest Repository* algorithm reduces the communication delay between clients and the repositories serving them. The *Divide and Conquer* strategy further improves the fidelity by reducing the average computational delay at repositories through coherency based categorization of client requests.

While the current approaches use push based dissemination, the solutions proposed can also be used with other mechanisms of data dissemination such as pull, adaptive combinations of push and pull [2], and leases [6]. Our future research shall focus on use of such alternative dissemination mechanisms as well as evaluation of our solutions in a real world Internet setting.

References

[1] S. Agrawal, K. Ramamritham, and S. Shah. Construction of a coherency preserving dynamic data dissemination network. Technical report, IIT Bombay, August 2004.

[2] M. Bhide, P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. J. Shenoy. Adaptive push pull: Disseminating dynamic web data. *IEEE transactions on Computers special issue on Quality of Service*, May 2002.

[3] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available server. *In Proceedings of the IEEE Computer Conference (COMPCON)*, March 1996.

[4] A. Fei, G. Pei, and L. Zhang. Measurements on delay and hop-count of the internet. *IEEE GLOBECOM'98 - Internet Mini-Conference*, 1998.

[5] B. Kao and H. G. Molina. Scheduling soft real time jobs over dual non-real time servers. *IEEE trans. on Parallel and Distributed Systems*, January 1996.

[6] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, and R. Tewari. Cooperative leases: Scalable consistency maintenance in content distribution networks. *In Proceedings of WWW10*, May 2002.

[7] C. Olston, B. Loo, and J. Widom. Adaptive precision setting for cached approximate values. *In Proceedings of the ACM SIGMOD Conference*, May 2001.

[8] C. Olston and J. Widom. Best effort cache synchronization with source cooperation. *In Procedings of the ACM SIGMOD Conference*, June 2002.

[9] K. Ramamritham. Real time databases. *Journal of Distributed and Parallel Databases*, 1(2):199–226, 1993.

[10] M. S. Raunak, P. J. Shenoy, P. Goyal, and K. Ramamritham. Implications of proxy caching for provisioning networks and servers. *Proceedings of ACM SiG-METRICS conference, pages 66-77*, 2000.

[11] S. Shah, K. Ramamritham, and S. Dharmarajan. An effi cient and resilient approach to fi ltering and disseminating streaming data. *Proceedings of the 29th conference on Very Large Data Bases*, September 2003.

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