Client Assignment in Content Dissemination Networks for Dynamic Data

Shetal Shah Krithi Ramamritham Chinya Ravishankar

Dept of Comp Science and Engg
Indian Institute of Technology Bombay
Mumbai, India

Comp Science and Engg Dept
University of California
Riverside, CA

Abstract

Consider a content distribution network consisting of a set of sources, repositories and clients where the sources and the repositories cooperate with each other for efficient dissemination of dynamic data. In this system, necessary changes are pushed from sources to repositories and from repositories to clients so that they are automatically informed about the changes of interest. Clients and repositories associate coherence requirements with a data item d, denoting the maximum permissible deviation of the value of d known to them from the value at the source. Given a list of <data item, coherence> served by each repository and a set of <client, data item, coherence> requests, we address the following problem: How do we assign clients to the repositories, so that the fidelity, that is, the degree to which client coherence requirements are met, is maximized?

In this paper, we first prove that the client assignment problem is NP-Hard. Given the closeness of the client-repository assignment problem and the matching problem in combinatorial optimization, we have tailored and studied two available solutions to the matching problem from the literature: (i) max-flow min-cost and (ii) stable-marriages. Our empirical results using real-world dynamic data show that the presence of coherence requirements adds a new dimension to the client-repository assignment problem. An interesting result is that in update intensive situations a better fidelity can be delivered to the clients by attempting to deliver data to some of the clients at a coherence lower than what they desire. A consequence of this observation is the necessity for quick adaptation of the delivered (vs. desired) data coherence with respect to the changes in the dynamics of the system. We develop techniques for such adaptation and show their impressive performance.

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

1 Introduction

Dynamic data is data which changes rapidly and unpredictably. Examples of such data are stock prices, information collected by sensors, network traffic, health monitoring information, etc. Increasingly more and more users are interested not only in monitoring dynamic data but also in using it for making timely on-line decisions. As a result, we need techniques to disseminate such data to users as efficiently as possible. Existing techniques for static data replicate the data closer to the users. This may however involve large communication and computational costs for fast changing data which makes the use of such techniques infeasible for dynamic data. To this end, we have been focusing on techniques to build an efficient, resilient and scalable content distribution network for dynamic data dissemination consisting of a set of cooperating sources and repositories, and clients as shown in Figure 1.

Clients request service by specifying the coherence requirements for the data items that they are interested in. The coherence requirement (c) associated with a data item d denotes the maximum permissible deviation of the value of d at the client from the value at the source. Let $S_d(t)$ and $U_d(t)$ denote the value of d at the source and the client, respectively, at time t. Then, to maintain coherence, we should have, $\forall t, |U_d(t) - S_d(t)| < c$. Typically, different clients will want different data items at different coherence requirements. For example, a user involved in exploiting exchange disparities in different markets or an on-line stock trader may impose stringent coherence requirements (e.g., the stock price reported

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 a less stringent coherence requirement. The content distribution network pushes changes to the data items to various repositories such that the client requests are met. Contrary to expectations, in [26] we showed that even push-based systems, unless designed carefully, can experience considerable loss in fidelity due to communication delays and computational overheads – for determining whether an update is relevant for a client and if so, pushing it to the client. These overheads, if not carefully managed and minimized, can lead to loss in data *fidelity*, that is, the degree to which the desired coherence requirements are met. Empirically, the fidelity f observed by a client for a data item is defined to be the total length of time for which the desired coherence requirement is met, normalized by the total length of the observation. Details of our content distribution network along with techniques to build such a network are given in [26].

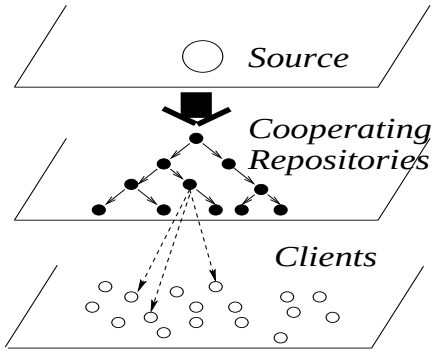


Figure 1: The Cooperative Repository Architecture

The issue that follows naturally from our previous work is that of serving the needs of the various clients that request service from the repositories in the network. Since clients obtain their data from repositories which form a cooperative content distribution network, achieving good data fidelity for clients necessitates solutions to two subproblems:

• *Assigning data to repositories*: Which repository should serve which data items, and at what coherence? The answer to this question should ideally take into account source and repository capabilities, rate of change of data items as well as clients' data and coherence needs.

• *Assigning clients to repositories*: Given a set of client requests, how do we assign them to the repositories such that the fidelity experienced by the clients is maximized? This should take into account data availability at repositories, the dynamics of the data, as well as overheads induced by the repositories to push updates to clients. These two problems are clearly inter-related. Furthermore, as clients can join or leave the network or their data needs may change with time, ideally, the two types of assignments need to be updated in a coordinated fashion for providing efficient service to the clients. There are two possible approaches for doing this - one in which the network dynamically adapts to the changing requirements and another, where periodically a snapshot of the clients' needs is taken and then the repository data needs as well as client-repository assignments are recomputed.

1.1 Problem: Assigning Clients to Repositories

We start with the assumption that the snapshot based approach has been used for assigning data to repositories; this allows us to focus on the client assignment problem. That is, we assume that the data needs of the repositories are already derived and examine ways to assign clients to the repositories.

Suppose, for a data item d, we have a set of n client requests and r repositories which serve d. In a typical content distribution network, $r << n$. In the absence of coherence requirements and computational delays, and in the presence of communication delays, we would like to *match* the clients to the repositories such that the overall communication delays are minimized. This can be looked upon as a many-to-one *weighted matching problem*[1] where one wants to find a matching of minimum weight, where the weights are the communication delays such that all the client requests are matched to the repositories.

The presence of coherence requirements and computational delays at the repositories complicate the problem considerably. As mentioned earlier, different clients will want a data item at different coherences. We say that the network is able to *satisfy* a client request if it can meet the coherence needs of the client. Note that a repository will be able to satisfy only those requests whose the coherence requirements are less stringent or equal to that at which the repository is providing service. Also, note that if we assign too many clients to a repository we could end up overloading the repository. This in turn could lead to larger update propagation delays at the repository, increasing the loss in fidelity. This calls for an adaptive approach that adapts both repository data needs as well as the coherence at which data is served to the clients.

1.2 Contributions: Solutions to the Client Assignment Problem

Firstly, we prove that *optimal client-repository assignment is NP-Hard* in Section 2. Since the problem is NP-Hard, we develop principled heuristics inspired by two well studied matching algorithms [29].

We can consider the repositories and the client requests as sets in the bipartite graph in which we seek a matching. An edge from a client request to a repository indicates that a client request can be served by this repository. As each repository will typically serve many

[1]In the literature, matching is one-to-one. Here we look at many-to-one matching.

clients, we need to do many-to-one matching. One way to solve a many-to-one matching problem is to convert the problem instance to that of a one-to-one matching problem by replicating the repositories appropriately and then solve using standard matching algorithms [23, 27]. This would however explode the problem size. Also, if an inappropriate replication of the repositories might result in the assignment of too many clients to a repository, thus overloading it. Another way to solve the problem is to use a *min-cost max-flow* algorithm to solve the assignment problem. Framing it as a min-cost flow instance not only allows us to control the number of assignments made to each repository but enables to do so without increasing the problem size. Here, the problem for all client requests for all data items can be solved at one go. This however does not quite take the repository overheads into account. This is due to the fact that it is not just the number of clients that determine the load at a repository but also the coherences of the clients assigned. Hence the repository load is known only after the assignment is made.

We deal with this conundrum as follows. We split the <client, data item> pairs into parts. We then run the max-flow min-cost subroutine on the parts one by one. After the i^{th} iteration, we know the load on each repository so far and this influences the cost function and the capacity on the edges for the next iteration. We can split the problem into parts based on either data items, coherence requirements or both. This way of incrementally solving the problem not only helps us to approximate the repository load better for future assignments but a judicious split also reduces the size and hence the overall running time. The exact details of this approach are presented in Section 3.

A completely different approach is to do bipartite graph matching by using *stable marriages*. The ability of the stable-marriage algorithm to accommodate coherence and cost-based preferences during the matching process makes it an good candidate for solving our problem. Consider a scenario, where a partial client-repository assignment has already been made. In such a scenario, remaining clients will prefer, say, repositories which are not heavily loaded. Repositories, in turn, would also prefer certain clients over the others. Coherence requirements of the clients, load on the repositories and the communication delays between the clients and the repositories play a role in calculating the preferences of the clients and the repositories for each other. Given these preferences, we can then use an algorithm for stable marriages to give us a stable assignment of clients to the repositories. We also propose an incremental version of this approach in Section 3.

We have thoroughly evaluated these client-repository assignment approaches using real world dynamic data and the results of our experimentation are presented in Section 3. An interesting result is that in highly update intensive situations a better fidelity can be delivered to the clients by attempting to deliver data to the clients *at a lower coherence than what the clients desire*. A consequence of this observation is the necessity for quick adaptation of the delivered (vs. desired) data coherence to the changes in the dynamics of the system. We develop techniques for such adaptation and show that *it is better to relax - by a large amount - the coherence offered to a few clients, than to relax - by a small amount, the coherence offered to all clients. That is, it might be better to be biased against* some *clients to improve overall fidelity for* all *clients.* We would like to mention that this is a known result in admission control mechanisms and we also observe this phenomenon here.

Related work is presented in Section 5.

2 NP-Hardness of Client Assignment

We now define the Client Assignment Problem, and show that it is NP-complete.

Client Assignment Problem:

Inputs: We are given the following:

• A system of s sources $\{S_1, S_2, \ldots, S_s\}$, r repositories $\{R_1, R_2, \ldots, R_r\}$, n clients $\{L_1, L_2, \ldots, L_n\}$, and k data items $\{d_1, d_2, \ldots, d_k\}$.

• For each repository R_i, the list of data items D^{R_i} it serves, and for each $d_j \in D^{R_i}$, the coherence level $c_j^{R_i}$ at which R_i serves d_j. Clearly, $|D^{R_i}| \leq k$.

• A set of client coherence requirement triples $\langle L_i, d_j, c_j^{L_i} \rangle$, stating that client L_i needs data item d_j at coherence level $c_j^{L_i}$.

• The distribution from which values of d_j are drawn, for each d_j.

• For each $\langle L_i, d_j, c_j^{L_i} \rangle$ triple, a number α_{ij} such that $0 < \alpha_{ij} < 1$.

• $\delta(n_1, n_2)$, denotes the communication delay between any two nodes n_1 and n_2 in the network. A node may be a source, a repository, or a client.

Each ⟨parent, dependent, data item⟩ triple is associated with a check delay and a push delay. Check delay is the time a repository takes to check if a dependent is interested in a particular update of the data item, and push delay is the delay to push the update to the dependent, respectively. For the rest of this section, by *parent* we mean a source or a repository and by *dependent* we mean a client or a repository, unless explicitly mentioned.

Question: Can every client L_i be assigned to some repository R_j such that for each client coherence tuple $\langle L_i, d_k, c_k^{L_i} \rangle$, we have $d_k \in D^{R_j}$ and $c_k^{R_j} \leq c_k^{L_i}$, and such that L_i receives d_k with fidelity at least α_{ik}?

Proof of NP-Hardness: To prove that the client assignment problem is NP-Hard, we reduce a well known NP-Complete problem, Partition [15], to the client assignment problem.

Partition:

Input: A set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements, each with a positive weight w_i and an integer $k \geq 2$.

Question: Can the elements of set S be partitioned into k parts such that the sum of the weights of the elements in each part is the same. That is, can S be split into k disjoint equal-weighted parts?

Given an instance for partition, we convert it to an instance for the client assignment problem by constructing a network of 1 source, k repositories, n clients and n data item as follows.

• Each repository R_j, $1 \leq j \leq k$ serves all the data items $d_1, d_2, \ldots, d_n$, serving d_i at a coherence $\frac{1}{w_i}$, $1 \leq i \leq n$.

• Client L_i requests data item d_i at coherence $\frac{1}{w_i}$, $1 \leq i \leq n$.

• The communication delay between any two nodes in the network is 0.

• The push delay for any data item from any repository to any client is μ where $\mu = \frac{k}{W}$ and $W = \sum_i w_i$. All check delays and all other push delays, including delays to push changes from the source to the repositories or from one repository to another, are 0. Hence, pushes from the source will be available immediately to all the repositories. Repositories will push data to clients after delay μ.

• The fidelity for all data items and clients is $\alpha = \frac{1-\epsilon}{2}$.

• All data items change deterministically, in accordance with the function shown in Figure 2. A change in value is initiated every 2 time units, and the value changes by 1 unit in exactly ϵ time units, where $\epsilon < \frac{\mu}{2k}$.

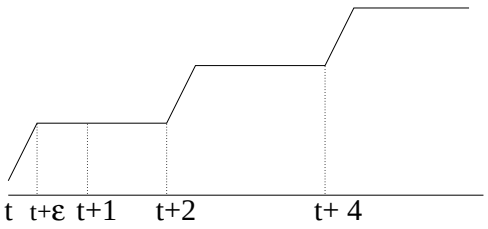


Figure 2: Distribution of Data Items

Each repository is required to serve d_i at coherence $\frac{1}{w_i}$, hence the source must send w_i updates to each repository when the value of d_i changes by one unit. Since changes occur at every 2 time units, the source sends w_i updates for d_i to all repositories at even time intervals. Every repository receives updates for all data items, but forwards only the updates for data items its clients are interested in. Such updates are called *useful* updates. Now, the check delays at all nodes are zero. The push delays between the source and the repositories are also zero. Additionally, the communication delays between any two nodes are zero and hence these updates reach the repositories instantaneously. The update now experiences a push delay from the repository to the client interested in it. Each useful update causes a delay of μ time units.

Claim: There is a client assignment such that the fidelity of every data item is at least $\frac{1-\epsilon}{2}$ *if and only if* there is a partition of S into k equally weighted parts.

Proof:

Forward direction: Suppose there is a partition of S into k equal partitions $P_1, P_2, \ldots, P_k$. The client assignment follows the partition. That is, if element s_i is in the partition P_j, then client L_i is assigned to repository R_j.

For the proof, we need only consider the time interval $[0, 2]$, since the process cycles every 2 seconds. In this interval, data item d_i changes by 1 unit in ϵ time units. That is, d_i induces w_i updates in ϵ time units, each time changing by $\frac{1}{w_i}$. Hence, the total number of updates at the source is $W = \sum_i w_i$, and the number of *useful* updates at any repository is $\frac{W}{k}$. Each repository gets all these updates in ϵ time units since the propagation delays are zero.

We claim that all updates reach the clients by time $1 + \epsilon$. The number of *useful* updates at any repository is $\frac{W}{k}$, so the time taken to push these updates to the clients is $\frac{W\mu}{k} = 1$ sec, since $\mu = \frac{k}{W}$ secs. Hence, all clients get all updates they are interested in within $1 + \epsilon$ time units. Since the data values at source and clients remain equal in the interval $[1 + \epsilon, 2]$, the fidelity of every data item across all clients is at least $\frac{1-\epsilon}{2}$.

Converse direction: Consider a client assignment such that the fidelity for every pair of the form (L_i, d_i) is at least $\frac{1-\epsilon}{2}$. We derive a partition of the corresponding set by making the weight of the elements in the partition equal to the number of *useful* updates w_{R_i} at repository R_i. If $w_{R_i} = \frac{W}{k}$ for all repositories R_i, we immediately have a solution for the partition problem.

Suppose $w_{R_i} > \frac{W}{k}$. Since the push delay for sending an update to a client is μ, R_i sends its last update to some client no earlier than $\mu \cdot w_{R_i}$ time units. Since $\frac{W\mu}{k} = 1$, we have $\mu \cdot w_{R_i} = \mu \cdot w_{R_i} - (\frac{W\mu}{k} - 1) = 1 + \mu(w_{R_i} - \frac{W}{k})$. However, $w_{R_i} - \frac{W}{k} = \frac{1}{k}(kw_{R_i} - W) \geq \frac{1}{k}$ since $w_{R_i} > \frac{W}{k}$. Consequently, $\mu \cdot w_{R_i} = 1 + \mu(w_{R_i} - \frac{W}{k}) \geq 1 + \frac{\mu}{k} > 1 + 2\epsilon$, since $\epsilon < \frac{\mu}{2k}$.

Under this situation, some data item d_k at client L_k was in sync with the source for less than $1 - \epsilon$ time units. Hence the fidelity for this (L_k, d_k) pair was less than $\frac{1-\epsilon}{2}$, which is a contradiction.

If $w_{R_i} < \frac{W}{k}$, then at least one other repository R_j gets $w_{R_j} > \frac{W}{k}$ updates. We repeat the argument above for R_j.

3 Client Assignment Using Min Cost Flows and Stable Marriage

Typically, an update to a data item of interest to a client is pushed to it via a network of repositories, and incurs communication and computational delays en route to the client, diminishing fidelity for that update. One way to improve fidelity is to lower the communication and computational delays in the network. We present two techniques for this purpose based on the Min Cost Flow

Problem and Stable Marriage Problem. These are well-studied combinatorial optimization problems, which we overview below. For details see [29].

3.1 Using Min-Cost Network Flows

In this model, we permit a client L_i to obtain each data item d_j of interest from a different repository. As mentioned earlier, assigning these $(L_i, d_j, c_j^{L_i})$ triples to repositories is a many-to-one weighted assignment problem, which can be solved using min-cost network flows. The cost of an assignment would ideally be a function of the communication delays, coherence requirement of the client and the repository and the load on the repository. The load on the repositories can be roughly balanced by placing a limit on the number of $(L_i, d_j, c_j^{L_i})$ triples that a repository can serve.

The Min-Cost Network Flow Problem: The input to a Min Cost Network Flow problem is a network of vertices and edges. Each edge has (i) a capacity which indicates the maximum flow the edge can support, and (ii) a cost associated per unit of flow. The vertices may also have some quantity of flow associated with them. The network has a source node and a sink node and the problem is to find a maximum flow of minimum cost in the network.

3.1.1 Client Assignment Using Network Flows

Figure 3 shows our approach to solving the client assignment using min cost network flows.

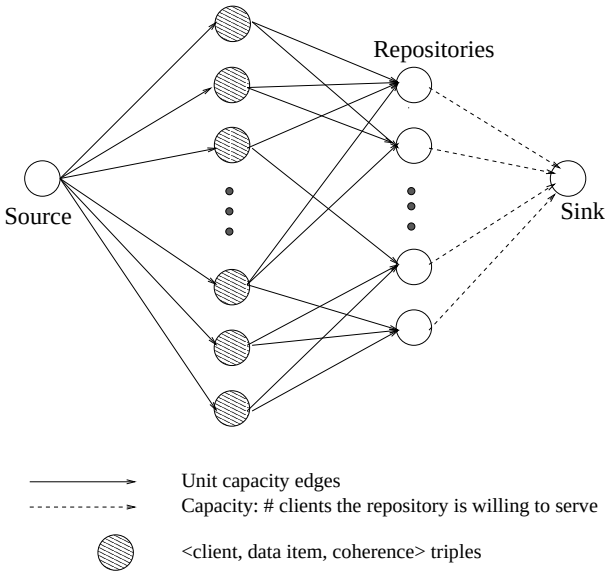


Figure 3: Min Cost Flow Formulation of the Client Assignment Problem

Vertices in the network:
Each repository and each $\langle L_i, d_j, c_j^{L_i} \rangle$ triple is represented by a vertex in the network. There is a source vertex from which the flow originates, and a sink vertex at which it is absorbed.

Edges in the network:

- An edge of unit capacity and unit cost exists from the source to every vertex representing a $\langle L_i, d_j, c_j^{L_i} \rangle$ triple.
- An edge exists from each $\langle L_i, d_j, c_j^{L_i} \rangle$ triple to every repository R_k that can serve d_j to L_i. The cost of this edge is a function of the delay between the repository R_k and the client L_i, the coherence value $c_j^{L_i}$ the client specifies for d_j, and the coherence value $c_j^{R_k}$ at which R_k can serve d_j. This function assigns lower costs to repositories for which $c_j^{R_k} < c_j^{L_i}$, and lowers costs further when $c_j^{L_i} - c_j^{R_k}$ is small. Such a cost function helps assign clients to repositories whose requirements are close to theirs, and hence optimizes the number of messages required for data dissemination.
- A zero-cost edge exists from every vertex representing a repository to the sink, with capacity equal to the maximum number of $\langle L_i, d_j, c_j^{L_i} \rangle$ triples that a repository can support.

We now find a min-cost flow in this network. It is well-known that the max flow in the network is integral when the capacities are positive integers. We ensure that each $\langle L_i, d_j, c_j^{L_i} \rangle$ triple is assigned to at most one repository by assigning unit capacity to the edge between the source and each $\langle L_i, d_j, c_j^{L_i} \rangle$ vertex, and to the edge between each $\langle L_i, d_j, c_j^{L_i} \rangle$ vertex and a repository that can serve the client. By ensuring sufficient capacity on the repository-sink edges, we obtain the solution for all $\langle L_i, d_j, c_j^{L_i} \rangle$ triples. This gives us an assignment of all the clients to the repositories.

3.1.2 Iterative Min Cost Network Flow

We have tried to roughly balance loads by bounding the capacities of each repository. Unfortunately, this approach is simplistic. The load due to a $\langle L_i, d_j, c_j^{L_i} \rangle$ triple depends on the coherence requirement. The load on a repository depends on the assignments made to the repository, and the assignments, in turn, depend on the repository loads. This is a chicken-and-egg problem. To get better estimate of the load, we split the problem into smaller units and solve by iteration. That is, rather than solve the network flow problem on all $\langle L_i, d_j, c_j^{L_i} \rangle$ triples simultaneously, we split the input into parts and then execute the network flow solver on each part. In each iteration, we estimate the load on the repositories due to the assignment made to them so far and use this to make further assignments.

We proceed as follows. We split the $\langle L_i, d_j, c_j^{L_i} \rangle$ triples into ranges, based on the coherence requirements. We run the network flow solver for successive ranges. Once we have assignments for a range, we calculate the approximate load on the repositories. The approximate load at a repository due to a triple $\langle L_i, d_j, c_j^{L_i} \rangle$ being as-

signed to it is given by $(c_j^{L_i})^{-2}$. Our reasoning is as follows: if the coherence is c, and a data item d is modelled as a random walk, then the number of updates generated is proportional to $\frac{1}{c^2}$ [26]. This gives us a rough indication of the load due to the assignment of a single client to the repository. This also implies that more stringent the coherence requirement for a data item, more is the load imposed on the repository. Due to this, we run the solver in the decreasing order of coherence stringency.

The approximate load is then used to determine the capacity of the repositories for the next range as follows.

Let there be r repositories and n $\langle L_i, d_j, c_j^{L_i} \rangle$ triples, each with coherence requirement α, to be assigned in the current iteration. The total load due to the assignments in this iteration is $\frac{n}{\alpha^2}$. Let the current load of a repository R_i be A_i and the average current load be A. We ask how many of these $\langle L_i, d_j, c_j^{L_i} \rangle$ triples should be assigned to each R_i. This should then be the capacity of the edge from R_i to the sink. We would like to do this to balance the load as much as possible.

Let t_i be the number of $\langle L_i, d_j, c_j^{L_i} \rangle$ triples assigned to R_i. Then the total load on R_i after assignment will be $A_i + t_i/\alpha^2$. To balance the load amongst the repositories, we equate this to the average load on a repository after assignment which is $A + \frac{n}{k\alpha^2}$. Hence we get $t_i = \frac{n}{k} + \alpha^2(A - A_i)$.

When t_i is negative, R_i has become more loaded than the other repositories, so we give a nominal capacity of 1 for the R_i-sink edge for that range.

3.2 Client Assignment via Stable Marriage

Consider n men and n women, and let each man and each woman rank all the members of the opposite sex by preference. The problem is to find a *stable marriage* between the sets of men and women, defined to be a pairing of men with women, in which there is no man m and no woman w such that m prefers w to his current partner and w prefers m to her current partner. Since we exclude pairs of men only or women only, this is equivalent to a bipartite graph matching.

3.2.1 The Gale-Shapely Algorithm

This is an iterative algorithm for the stable marriage problem. In the first iteration, each man proposes to the woman he ranks highest, and each woman rejects every proposal except the one from the man she ranks highest, whom she keeps pending. In the next iteration, each rejected man proposes to the women next in their preference list. Each woman again ranks the new proposals and any proposal she has pending, and keeps the one she ranks highest pending, rejecting the rest. This process continues till no further rejections take place. This algorithm is known to terminate and to provide a stable marriage in $O(n^2)$ time using efficient data structures.

3.2.2 Client Assignment Using Stable Marriages

We can consider the repositories and the $(L_i, d_j, c_j^{L_i})$ triples as the sets in the bipartite graph in which we seek a matching. Each $(L_i, d_j, c_j^{L_i})$ triple ranks the repositories using a suitable preference function, based on factors such as the communication delays, load on the repositories and coherence requirements. The repositories can similarly rank the $(L_i, d_j, c_j^{L_i})$ triples. If there are n repositories, we can, in each iteration, assign some n of the $(L_i, d_j, c_j^{L_i})$ triples to these repositories using the stable marriage algorithm. This approach has the advantage that the ranking function in each iteration can take into account the the load at each repository due to the $(L_i, d_j, c_j^{L_i})$ triples already assigned. Its chief disadvantage is that it is based only on the ordering of preferences, rather than actual parameter values.

We consider the data items one at a time. The clients interested in some data item d represent the n men in the stable marriage problem and the repositories which serve d represent the r women. Since ours is a many-to-one assignment problem, the marriage is polyandrous. The number of clients that a repository can accept is a function of the number of clients and the repositories.

Once the preferences of the clients and the repositories are calculated, we assume they remain unchanged till the next snapshot. Clients preferences are based on coherence requirements and repository loads, and prefer lightly loaded repositories with coherence requirements equal to (or more stringent than) the clients' requirements. Given a client C which wants data item d at coherence c_C and repository P which serves d at coherence c_P, the heuristic to calculate the preference is given by: $P_{current\_load} * \frac{c_C}{c_P}\ if(c_P \leq c_C)$ and $P_{current\_load} * \frac{c_P}{c_C} * 100\ if(c_C < c_P)$. The smaller the preference value, the more preferred is the repository. Since the client would prefer to be served by a repository whose requirements are more stringent than its own, we multiply the preference factor by a large constant if $c_P > c_C$ This way, repositories with requirements less stringent than the clients' will get a lower preference value. Of the repositories that can serve the client, we would like to choose a less loaded repository for the service. The current load at the repository is calculated as a function of the coherence requirements of the clients assigned to it as given in Section 3.1.2.

The repositories rank the clients based on the communication delay between the repository and clients and the coherence requirement of the client. A repository prefers clients whose coherence specifications are close to, but less stringent than its own. They also prefer clients to whom the communication delays are lower.

Once these preference values are calculated, the Gale-Shapley algorithm is executed and clients assigned to the repositories in accordance with the resulting stable marriage.

Some notes and optimizations:

If a repository with loose coherence requirements is matched to a client with stringent coherence requirements, the client will miss a lot of updates. To avoid such mismatches, the clients and the repositories are split into two groups - one where stringent coherence and one with loose coherence. For each data item, the algorithm is executed once for each group, ensuring that clients get repositories in the same group. One can determine the range of stringent and loose coherences from the coherence requirements served by the repositories. This optimization alone improved the fidelity obtained by 20%.

3.3 Augmentation

It is not always possible to find a repository satisfying a clients' needs. For e.g., a client L may be interested in a data item d with coherence requirement 0.01 but all the repositories in the network serving d are serving at a coherence of 0.02 or above. In this case, we can do one of the following:

1. *Best-Effort Mapping:* In this we try and find the repository whose requirements are as close to that of the clients. While the client will not get every single change that (s)he is interested in, this is the best that the network can provide. In the above example, L will be served with a coherence of 0.02. The client may experience a higher loss in fidelity but it will continue to get best possible service from the network.

2. *Augmentation:* In this, we augment the requirements of the repository chosen for the clients by best effort mapping. For a data item, the coherence served by a repository is then the most stringent of that of the clients assigned to it. For example, in the above example, the repository serving L will now start serving d with a coherence of 0.01.

4 Experimental Methodology and Results

We now present the experimental methodology and then results for the performance evaluation of client assignment techniques.

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 1000 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 prices observed in the 10000 values polled during the indicated *Time Interval* on the given $Date$ in Jan/Feb 2002.) As we obtained 10000 polls in roughly 3 hrs, we were able to obtain a new data value approximately once per second(3*3600/10000). 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*
MSFT	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 and Clients – Data and Coherency characteristics: Each repository and client requests a subset of data items, with a particular data item chosen with 50% probability. A coherence requirement c is associated with each of the chosen data items. We use different mixes of data coherence, the c's associated with the data in a client or repository are a mix of stringent tolerances (varying from 0.01 to 0.09) and less stringent tolerances (varying from 0.1 to 0.99). T% of the data items have stringent coherence requirements (the remaining $(100 - T)$%, of data items have less stringent coherence requirements). To give an indication of the number of client requests in the network: if the number of clients is, say 100, and number of data items is, say 200, then the total number of the client requests will be approximately $\frac{100*200}{2} = 10,000$.

Physical Network – topology and delays: The physical network consists of nodes (routers, repositories, sources and clients) and links. The router topology was generated using BRITE (http://www.cs.bu.edu/brite). Once the router topology was generated we randomly placed the repositories, clients and the sources in the same plane as that of the routers and connected each to the closest router. For each repository and client, the data items of interest were first generated and then coherences were chosen from the desired range.

Our experiments use node-node communication delays derived from a heavy tailed Pareto [24] 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 [13]. Depending on the nature of the queries, the check delay (see Section 3.1.2) can be short for a simple check to the tune of a few tens of milliseconds for some complex query processing [10, 19]. We derive the checking delay for each data item for each repository from a heavy tailed Pareto distribution with $\bar{x}$ as 5 ms and x_1 1 ms. The average checking delay was around 4 ms. The push delay was also derived using a Pareto

distribution with $\bar{x}$ as 1 ms and x_1 0.125ms. We also experimented with other check and push delays.

Metrics: The key metric for our experiments is the loss in fidelity of the data. Fidelity is the degree to which a user's desired coherency requirements are met and is measured as the total length of time for which the coherence requirements are met normalized by the total length of the observations. The fidelity of a client is the mean fidelity over all data items requested at that repository, while the overall fidelity of the system is the mean fidelity of all clients. The loss in fidelity is simply $(100\% - \text{fidelity})$. Clearly, the lower this value, the better the overall performance of a dissemination algorithm.

Min Cost Network Flow Solver: The maximum flow of minimum cost in the network was obtained using an algorithm by Bertsekas et al [6] who use a relaxation method to solve the network flow problem. The solver that we used was RelaxIV; the code for which was obtained from [1].

Iterative Min Cost Flows: In iterative min cost flows, we split the coherences required into ranges and execute the network flow solver on each range. For the results shown, the ranges used were $\{0.01 - 0.03, 0.03 - 0.07, 0.07 - 0.2, 0.2 - 1\}$.

An On-line Global Heuristic for Comparison: The techniques developed in this paper are compared amongst themselves and also with an on-line global heuristic [4]: For each data item d, a selector node in the network maintains information about (a) the repositories which serve d and their coherence requirements, (b) the number of clients already assigned to each repository and (c) the communication delays between the nodes in the network. When a client needs service from the network, for each data item, the corresponding selector tries to find a repository such that (a) the coherence requirement of the client is satisfied and (b) the sum of the communication and computation delays is minimized. The computational delay at repository R is approximated by the number of clients served by R. We present results of two versions of this heuristic: (i) *GH*: In this heuristic, if no repository can meet the requirements of the client, then the source serves the client. (ii) *GH-be*: This does best effort mapping as explained in Section 3.3.

4.1 BaseLine Results

Figure 4 presents the initial results of our experimentation. Figure 4(a) shows the results for T=20, i.e., when 20% of the data item requests are stringent; also only 20% of the data items served by a repository are served with stringent coherences. In a network of 20 repositories, for T=20, only about 2 repositories will serve the data item from the stringent range which is $0.01 - 0.09$ for our experiments. Our data assignment randomly assigns a value to serve from the range. Hence, it is quite likely that for some data items, we may have client requests which are more stringent than those served by the repositories [2]. Due to this, for $T = 20$, the percentage of stringent requests that can be satisfied by the network is smaller than at $T = 50$ or $T = 80$. Since, GH-be, Min Cost Flows (MCF), and Stable Marriages(SM) do best effort mapping, for the $T = 20$ case, the loss in fidelity is high even for a small number of requests.

The Global Heuristic (GH) assigns the unsatisfied clients to the source. When number of clients is small, this does well but as the number of clients increases, the source gets overloaded and the heuristic starts to degrade rapidly. In fact, beyond 200 clients, the loss in fidelity is considerably higher than that offered by the other techniques. Hence, we do not show the results of GH beyond a point - the loss is much higher than 30%.

In Figure 4, we see that GH-be and MCF and SM are fairly comparable. On further experimentation, we found that the stringent requests are the ones which determine the fidelity observed by the network. If we cannot meet these requests, then the observed loss in fidelity could be high (as seen in the T=20 case). At the same time, they also add considerably to the load and hence, if not assigned properly, could also result in high loss in fidelity. In the networks above, for a given data item, about 2-8 repositories were serving the data item at a stringent coherence. The best effort based approaches tend to distribute the stringent requests amongst these repositories. Hence, as long as the best effort techniques do some simple load balancing, these techniques could be considered to be comparable at least for small to medium number of requests.

The results in Figure 4 also show that the fidelity observed by the clients is dependent on the coherence of the data at the repositories. We can see that if the network can satisfy all the requests, as given by the curves in MCF_aug and SM_aug, then the loss in fidelity due to data dissemination is very low. This would lead one to the conclusion that using augmentation would reduce the dependence of fidelity on the data assignment so that for even a naive data assignment, one can enjoy reasonably high fidelity. In Section 4.3, we show that this is not always true and also discuss ways of reducing this dependence.

4.2 MCF is the Preferred Approach

To compare the techniques in scenarios where the clients can choose from a larger set of repositories, we experimented with topologies where all the repositories serve all the data items at the most stringent coherence required. With this we focus on that part of a network where a large number of clients have more repositories to choose from. The results of this experimentation are in Figure 5. Initially, a for small number of requests, the techniques are comparable but as the number of requests increases, the best effort techniques start to perform bet-

[2] We feel that, even in a real network, one cannot always ensure that *all* the client requests will be satisfied by the network.

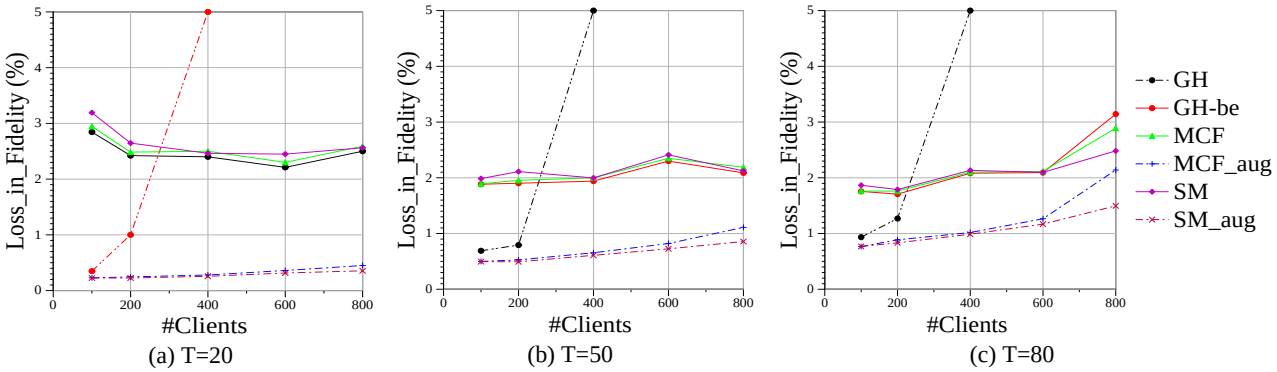


Figure 4: BaseLine Results: 1 source, 20 repositories, 200 data items.

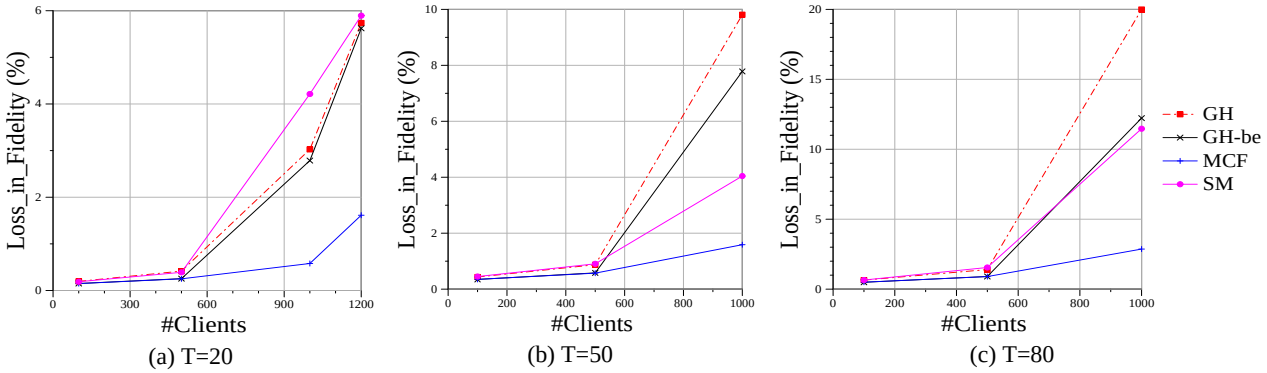


Figure 5: MCF Performs Best:1 source, 10 repositories, 50 data items

ter than GH with MCF doing better than GH-be and SM. In fact, in some cases, MCF did better than GH-be by as much as a factor of 9 in the resultant fidelity. We further investigated this by calculating the average queueing delay at the repositories. As we have mentioned earlier, we feel that a way to improve fidelity offered by the network is to reduce the delays in the system. We noticed that after a certain point the queueing delays for all the best effort heuristics tend increase exponentially but the rise of the delays for MCF is much slower than that of GH-be. We feel this is because MCF explicitly takes repository load into account (Section 3.1.2) as opposed to GH-be. Note that for these graphs, augmentation performs similar to the corresponding best effort approach as the network can meet the needs of the clients and hence separate curves are not shown for MCF_aug and SM_aug.

From Figures 4 and 5, we can see that the solution obtained from stable marriages can do fairly well, in fact sometimes even better than the other techniques. However, we prefer to use MCF rather than SM as we found SM to be very sensitive to the client-repository preference orderings of each other. We found that a different ordering resulted in an increase in the loss of fidelity obtained from 0.89 to 14.2 - a factor of almost 14.

4.3 Feedback driven Adaptive Augmentation

The curves in Figure 4 indicate that one of ways of achieving high fidelity is to augment the repositories to meet the client needs. However, Figure 6 shows that this can be counter productive, especially when the load is high. If the number of requests and hence the load on the network increases (say, after number of clients exceeds 400) then MCF_aug starts to perform worse than MCF, the approach without augmentation. We discuss the curves marked with adapt later in this section. The results indicate that for large number of requests, one should either not augment or do selective augmentation. We explored the issue of selective augmentation. There are two possible ways of doing this.

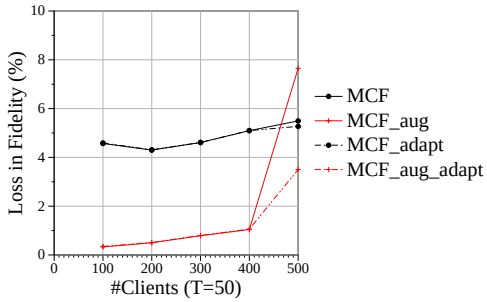
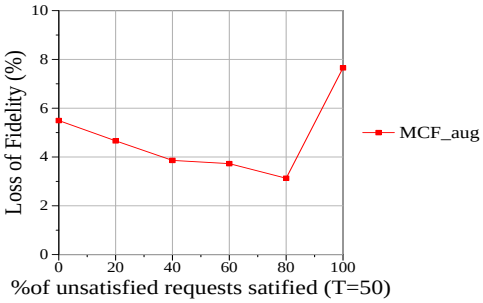
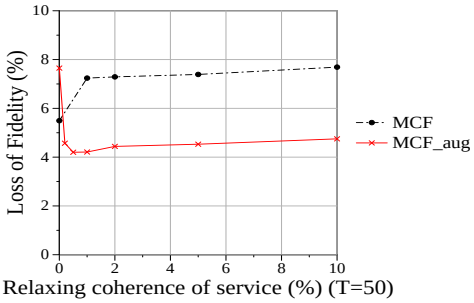


Figure 6: Client Assignment with Adaptation



(a) Augment to satisfy only some of the unsatisfied requests



(b) Relaxing the coherence of service for all the requests

Figure 7: Towards Adaptive Augmentation (1 source, 10 repositories, 200 data items, 600 clients

1. *A Biased Approach*: In this, we augment only a fraction of those requests that need augmentation. Essentially, in this approach, some of the clients will not be served at their desired coherences but the rest will be. The difference in the desired and served coherence might be high, sometimes even greater than than 100%, as explained in Section 3.3. Figure 7(a) shows the result of this experimentation. The total number of client requests was about $50,000$ and of these about 15% needed augmentation. The X axis shows the percentage of these requests that were augmented. The leftmost point in the curve is the fidelity offered by best effort (MCF) and the rightmost point is MCF_aug which offers 100% augmentation.

2. *A Fair Approach*: In this approach, we serve all the clients at a slightly looser coherence. In this, we relaxed the coherence (i.e., made it looser) at which the repositories served data for *all* the clients. We present these results in figure 7(b). We relaxed the coherence of service provided from 0% to 100% and present the results for both MCF and MCF_aug. Figure 7(b) only shows the trend from 0% to 10%, for the remaining values a linear increase in the loss in fidelity was observed.

In Figure 7(b), we see that the fidelity loss with MCF increases as we relax the coherence of service provided. This is because of the best effort service of MCF: as the coherence of service becomes looser, the fidelity drops. On the other hand, in both 7(a) and (b), MCF_aug shows an interesting behaviour. As we relax the coherence of service or reduce the percentage of requests that are augmented, the fidelity actually improves before dropping. This brings us to an interesting conclusion: *it might be better to serve the clients at a looser coherence than required by the clients, as the number of clients increases in the network.* A more stringent coherence requirement might increase the load at the source and the repositories and this might result in a higher loss in the fidelity than serving a client request at a looser coherence requirement. In fact, source overload is a large contributing factor to the loss in fidelity. We observed that for a 4 server, 800 data items, 100 clients and $T = 80\%$ topology, the loss in fidelity was about 4.5% - out of this about 90% was due to the delays at the source. Only 10% of the loss was due to the rest of the network.

The lowest loss in fidelity observed in Figure 7(a) for MCF_aug is less than that observed in Figure 7(b) (3.12 vs 4.2). Note that in Figure 7(a) some of the clients are served at a coherence which is at least than 100% of the requested coherence. The loss in fidelity however is lower than that obtained by increasing the coherence of service provided to all by 1%. This brings us to another interesting observation: *it is better to relax - by a large amount - the coherence offered to a few clients than to relax - by a small amount, the coherence offered to all clients. That is, it might be better to be biased against some clients to improve overall fidelity for* all *clients.*

The observations mentioned above lead us towards a feedback based adaptation for providing service to the clients.

1. For every data item, the source maintains a list of unique coherences that it serves in the network and the number of clients being served for each coherence. For each data item, this list is stored in the ascending order of coherences.

2. Once the clients are assigned to the repositories, during the actual dissemination of data in the network, if the source or repository observes that the queueing delays are large (i.e., the delays are more than a threshold, say th_1) then it informs the source to relax the coherences for some of the clients.

3. For each data item d_i, the source then takes the most stringent coherence c_i^1 and of these selects the $< d_i, c_i^1 >$ pair that has the minimum number of clients n_i. The network will now not provide service to these clients at that coherence but the next in the list, c_i^2. We add n_i to the number of clients requesting c_i^2.

4. The source now waits for a suitable time interval for the queueing delays to reduce. If the delays fall below the threshold, the node now informs the source

to stop relaxing the coherences. Else, the previous step is repeated.

5. If the queueing delays at the node become fairly small, we then start resuming service at the previous coherence - in the Last Out First In fashion. The source similarly waits for some time for the delays to stabilize, before resuming service of more requests, till they reach another threshold - th_2.

Figure 6 presents our results using this strategy. We set th_1 as 400 ms and th_2 as 40 ms in this experiment. We also experimented with different thresholds varying from 1 sec to 100 ms - (note that our data items change every 1-2 seconds) and observed a gain in the fidelity obtained for all these values. For 500 clients, the loss in fidelity for MCF_aug augmentation drops from 7.8% to 3.56%, by more than half!

In summary, we can state that an approach that combines the (a) philosophy of Min Cost Max Flow Approach, (b) augmentation and (c) adaptive coherence setting, works well even in situations when the number of client requests is large or the data is very dynamic.

5 Related Work

Push-based dissemination techniques that have been developed include broadcast disks [3], publish/subscribe applications [20, 5], web-based push caching [17], and speculative dissemination [7].

The design of coherence mechanisms for web workloads has also received significant attention recently. Proposed techniques include strong and weak consistency [19] and the leases approach [12, 30]. Our contributions in this area lie in the definition of coherence in combination with the fidelity requirements of users. Coherency maintenance has also been studied for cooperative web caching in [31, 28, 30]. 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 [18] where push-based invalidation and dependence graphs are employed to determine where to push invalidates and when. Scalability can be improved by adjusting the coherence requirements of data items [32]. The difference between these approaches and ours is that repositories don't cooperate with one another to maintain coherence.

Work on scalable and available replicated servers [32], and distributed servers [11] are related to our goals. [32] 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, and dissemination of dynamic data in a network is based on the coherence requirements of the clients. 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.

Mechanisms for disseminating fast changing documents using multicast-based push has been studied in [25]. 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 coherence tolerances. Multicast tree construction algorithms in the context of application-level multicast have been studied in [14]. 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 coherence of dynamic data.

Several research groups and startup companies have designed adaptive techniques for web workloads [9, 2]. But as far as we know, these efforts have not focused on distributing very fast changing content through their networks, instead, handling highly dynamic data at the server end. Our approaches are motivated by the goal of offloading this work to repositories at the edge of the network.

The concept of approximate data at the users is studied in [22, 21]; the approach focuses on pushing individual data items directly to clients, based on client coherence requirements. We believe that the two approaches are complementary since our approaches to cooperative repository based dissemination can be used with their basic source-client based dissemination.

Our work can be seen as providing support for executing continuous queries over dynamically changing data [19, 10]. Continuous queries in the Conquer system [19] are tailored for heterogeneous data, rather than for real time data, and uses a disk-based database as its back end. NiagraCQ [10] 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). [16] looks at pull-based approaches to satisfy a data aggregator needs. The work from this paper can used to determine data coherence needs to execute queries over dynamic data so that query results satisfy coherence requirements.

6 Conclusions and Future Work

In this paper we have discussed the client assignment problem in a content distribution network for dynamic data. Besides proving that the client assignment problem is NP-Hard, we have cast the client assignment problem as an application of two well studied algorithms from combinatorial optimization. The techniques we have developed can be used periodically to (re)assign clients to repositories.

We are currently investigating the mechanisms mentioned here in a real network setting, as exemplified by PlanetLab.

Whereas our approach uses push-based dissemination, other dissemination mechanisms such as adaptive pull [16, 33] or adaptive combinations of push and pull [8], could be used to disseminate data through our repository overlay network. The use of such alternative dissemination mechanisms is the subject of future research.

Acknowledgements

We would like to thank Shweta Agrawal for her help in the initial stages of this work.

References

[1] RelaxIV: www.di.unipi.it/di/groups/optimize/ORGroup.html.

[2] S. Gribble A. Fox, Y. Chawate and E. Brewer. Adapting to network and client variations using active proxies: Lessons and perspectives. *IEEE Personal Communications*, August 1998.

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

[4] S. Agrawal, K. Ramamritham, and S. Shah. Construction of a coherency preserving dynamic data dissemination network. *IEEE International Real-time Systems Symposium*, 2004.

[5] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *International Conference on Distributed Computing System*, 1999.

[6] D. Bertsekas and P. Tseng. Relaxation methods for minimum cost ordinary and generalized network flow problems. *Operation Research Journal*, 36:93–114, 1988.

[7] 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.

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

[9] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[10] 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.

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

[12] V. Duvvuri, P. Shenoy, and R. Tewari. Adaptive leases: A strong consistency mechanism for the world wide web. *Infocom*, March 2000.

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

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

[15] M. Garey and D. Johnson. *Computer and Intractability*. W. H. Freeman and Company, 1979.

[16] R. Gupta, A. Puri, and K. Ramamritham. Executing incoherency bounded continuous queries at web data aggregators. In *WWW*, 2005.

[17] J. Gwertzman and M. Seltzer. The case for geographical push caching. In *Proceedings of the Fifth Annual Workshop on Hot Operating Systems*, 1995.

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

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

[20] G. 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.

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

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

[23] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity*. Prentice Hall, 1982.

[24] M. Raunak, P. 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.

[25] P. Rodriguez, K. Ross, and E. Biersack. Improving the WWW: caching or multicast? *Computer Networks and ISDN Systems*, 1998.

[26] S. Shah, K. Ramamritham, and P. Shenoy. Resilient and coherency preserving dissemination of dynamic data using cooperating peers. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):799–812, July 2004.

[27] R. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[28] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Beyond hierarchies: Design considerations for distributed caching on the internet. *In IEEE International Conference on Distributed Computing Systems*, 1999.

[29] Douglas West. *Introduction to Graph Theory*. Prentice-Hall, India, 1999.

[30] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Hierarchical cache consistency in a WAN. In *USENIX Symposium on Internet Technologies and Systems*, 1999.

[31] 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.

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

[33] S. Zhu and C. Ravishankar. Stochastic consistency, and scalable pull-based caching for erratic data sources. In *Proceedings of the 30th Conference on VLDB*, August 2004.