Efficiently Maintaining Stock Portfolios Up-To-Date On The Web

Manish Bhide Krithi Ramamritham Prashant Shenoy†
Laboratory for Intelligent Internet Research †Dept of Computer Science
Department of Computer Science and Engineering University of Massachusetts
Indian Institute of Technology Bombay Amherst, MA 01003
Mumbai, India 400076 USA
{manishb,krithi}@cse.iitb.ac.in shenoy@cs.umass.edu

Abstract

Consider a continuous query where a user wants to be informed when the net worth of his/her stock portfolio changes by more than a specified threshold. In this paper we develop a data dissemination technique for the Web where (a) such queries access data from multiple sources and (b) the HTTP protocol – which is inherently pull based – is used for accessing the sources. Key challenges in supporting such queries – which also arise in a diverse set of contexts including monitoring of patients, network traffic, and experiments – lie in meeting users' consistency requirements while minimizing network and server overheads, without the loss of fidelity in the responses provided to users. We also show the superior performance of our technique when compared to alternatives based on periodic independent polling of the sources.

Keywords: Dynamic Data, Temporal Coherency, Scalability, Resiliency, World Wide Web, Data Dissemination, Push, Pull

1 Introduction

Many popular financial sites cater to the dissemination of stock prices to users (e.g., finance.yahoo.com). Users accessing these sites typically track a group of stocks that are of interest (referred to as their *portfolio*). Any query on the portfolio, such as the computation of the total value of the portfolio, requires these sites to first query a stock quote server to determine the most up-to-date value of each stock in the portfolio. Since financial data such as stock prices vary with time, such sites do not cache such information and prefer to poll the server for the most recent value of each data item. These sites are further hampered by the lack of effective mechanisms to maintain coherency of a cached, time-varying data item with its server version. A server-based approach is not viable because it not scalable from the perspective of supporting a large number of users. To alleviate this drawback, in this paper, we consider an approach where time-varying data items such as stock prices are cached either at a proxy or by financial web sites. We then propose techniques to maintain coherency of cached data with the origin servers. Specifically, we consider coherency for a class of user queries that we refer to as *portfolio queries* where we attempt to track the total value of a portfolio with a specified accuracy. Whereas developed in the context of financial information such as stock prices, our approach can also be used to answer similar queries on any kind of dynamic data (e.g., traffic conditions, sensor data). Our technique is entirely pull based and it requires no special support from the server. Hence it can be readily used with any server adhering to the HTTP protocol.

In the rest of the section, we (a) describe the problem of temporal coherency related to portfolio queries, (b) show the limitations imposed by HTTP protocol to process such queries and then outline the key contributions of

this paper, namely, a technique to minimize the network overhead for such kind of queries, without the loss of fidelity in the responses provided to the users.

1.1 The problem of maintaining temporal coherency for portfolio queries

Suppose users obtain their time-varying data from a proxy cache. The caches that are deployed to improve user response times [6, 7] must track such dynamically changing data so as to provide users with temporally coherent information. To maintain coherency of the cached data, each cached item must be periodically refreshed with the copy at the server. In the context of the HTTP protocol, which is pull-based, several techniques, such as, *Time To Live (TTL)* [1] and client polling [3] have been developed to to maintain coherency of cached data. In this paper we demonstrate that, in the case of portfolio queries, using these techniques for individually retrieving each of the data items is not efficient - *additional mechanisms must be employed which exploit the fact that the stocks constituting the portfolio form a semantic unit*.

Typically a user is interested in changes to his/her portfolio that exceed a certain threshold. We call this threshold the *coherency requirement* (cr). To illustrate, a user may have 100 stocks of company A and 200 stocks of company B and he may be interested in monitoring changes greater than 100$ in the portfolio. This 100$ denotes the maximum permissible deviation of the value of the portfolio known to the user compared to its actual value at the server in the beginning. So, the proxy from which a user's needs are served [1] need not keep track of each and every change in the individual stock prices. Still, since the stock portfolio's current value is determined from the value *pull*ed for each of the stocks in the portfolio, the technical question we answer in this paper is: *How should one derive the coherency requirement of each of the stocks constituting a portfolio, given the coherency requirement for the portfolio?*. The coherency requirement associated with a data item depends on (a) the overall contribution of the stock to the portfolio (b) the coherency requirement for the portfolio.

As shown in Figure 1, let $S(t)$, $P(t)$ and $C(t)$ denote the value of an individual stock at the server, proxy cache and the user, respectively. Then, to maintain coherency we should have $|C(t) - S(t)| \leq tcr$ for each of the stocks, where tcr is the constraint allocated to each individual stock. Further in the realm of portfolio queries the following equation should also be satisfied

$$if \sum_{all\ stocks} |S(t) \times N(t) - S(t^{'}) \times N(t)| \geq threshold$$

then

$$\sum_{all\ stocks} |C(t) \times N(t) - S(t^{'}) \times N(t)| \geq threshold$$

where

- $S(t)$ is the current stock price at the server
- $S(t^{'})$ is the initial stock price at the server
- $C(t)$ is the current stock price at the client
- $N(t)$ is the number of stocks, and

[1] Techniques for disseminating data from proxies to end-users are not considered here, since resources such as network bandwidth are often plentiful on the proxy-user data path.

- $threshold$ specifies the coherency requirement for the stock portfolio.

It may happen that due to loss of coherency, the user may perceive the wrong state for his/her portfolio. There are two classes of misperceptions: (i) *false positive*, when a user thinks that the threshold has been exceeded, whereas it is well within the threshold, and (ii) *false negative*, when the portfolio changes have actually exceeded the threshold but the user is unaware of this. In some sense, the latter could be more harmful than the former because in the case of a *false positive* the proxy can always verify the value of the portfolio by polling all the stocks. But no such solution exists in case of false negatives. So it is important to avoid or at least minimize false negatives.

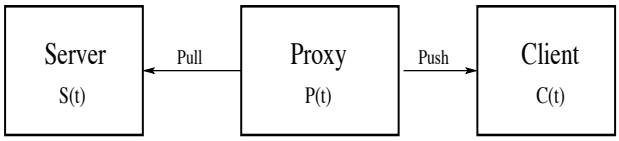


Figure 1: Coherency Requirement

To quantify the misperceptions, we define *fidelity* of the data seen by users to be the total length of time that the above inequality holds (normalized by the total length of the observations). In addition to the fidelity, another measure of evaluating the algorithm is to measure the number of false positives and false negatives that have occurred. In other words, the fidelity is the total length of time that the user correctly knew that the threshold was reached (normalized by the total length of the observation) i.e., the total time duration when there were neither false positives nor false negatives.

1.2 Problems with HTTP and the need for the adaptive distribution of constraints among stocks in a portfolio

The HTTP protocol is inherently pull based. The proxies need to frequently pull the data from the server based on the dynamics of the data and the coherency requirement of the user. The proxy can use the if-modified-since HTTP requests [9] to obtain any changes in the data value at the server. Very few servers provide push support and this service often comes at a price, namely space and computation overheads at the server. Using push, the number of false positives and false negatives can be made small, barring network and computational delays. However, the servers have to be re-engineered to deal with portfolio queries. Our goal is to work within the standard HTTP protocol and hence use pulls intelligently to maintain the coherence of portfolios within user allowed thresholds values.

As outlined earlier, the coherency requirement imposed on each of the data items constituting a portfolio will depend on (a) the overall contribution of the data item to the portfolio and (b) the coherency requirement of the portfolio itself. A very naive way to assign crs to data items would be to assign a small (i.e., stringiest) coherency to a data item that makes the greatest contribution to the portfolio (and hence such a data item will be polled more often). But the problem with this approach is that it does not consider the dynamics of the data items. Depending on the way the values of the various data items are changing the crs of each of the data items can be manipulated, thereby saving a lot of network overhead. For example, consider a portfolio consisting of 100 stocks of company A and 400 stocks of company B. Now if the company B stock hasn't changed for a long time and the company A stock sees a lot of trading and hence its price fluctuates a lot, then the cr of company A should be reduced and that

of company B should be increased. This will help in reducing the false negatives and it will also help in saving a lot of unnecessary pulls of the company B stock. Hence there is a need to judiciously assign the cr to each of the data items depending on the dynamics of the data. How to achieve this is the crux of the paper.

But this begs the following question: Given the cr of a single data item, how and when should it be polled so that the cr of this data item is maintained? We answer this question first and then move on to the problem of assigning crs to the entities constituting the portfolio. Finally, we provide experimental evidence for the efficiency of the proposed approach.

2 The Adaptive TTR Algorithm

To achieve the cr of a data item, the proxy computes a *Time To Refresh (TTR)* for the data item. The TTR denotes the next time that the proxy should poll the server so as to refresh the data item if it has changed in the interim. The success of the pull-based technique hinges on the accurate estimation of the TTR value. The *Adaptive TTR Algorithm*[2], reviewed next, is used to calculate the TTR associated with each data item.

This algorithm allows the proxy to adaptively vary the TTR value based on the rate of change of the data item. The TTR decreases dynamically when a data item starts changing rapidly and increases when changes are smaller and slower. To achieve this objective, the Adaptive TTR approach takes into account

- static bounds so that TTR values are not set too high or too low,
- the most rapid changes that have occurred so far and
- most recent changes to the polled data.

The Adaptive TTR is computed as

$$TTR_{adaptive} = Max(TTR_{min}, Min(TTR_{max}, TTR_{dynamic}, TTR_{mr'}))$$

where (TTR_{min}, TTR_{max}) denote the window within which the TTR values are bound. If the source data does not change too often, the TTR values will tend to be larger and hence might miss out many changes. To fix this problem, the bounding window is placed on the TTR values.

$TTR_{dynamic}$ is a learning based TTR estimate founded on the assumption that the dynamics of the last few recent changes are likely to be reflective of changes in the near future.

$$TTR_{dynamic} = ((w \times TTR_{estimate}) + ((1 - w) \times TTR_{latest}))$$

where

- $TTR_{estimate}$ is the an estimate of the TTR value, based on the most recent change to the data.

$$TTR_{estimate} = \frac{TTR_{latest}}{|D_{latest} - D_{penultimate}|} \times c$$

here D_{latest} denotes the latest data value obtained from the server and $D_{penultimate}$ specifies the previous data value obtained from the server.

- weight w $(0.5 \leq w \leq 1)$ is adjusted by the system so that we have the recency effect, i.e., more recent changes affect the new TTR more than the older changes.

$TTR_{mr'}$ in the above equation is chosen as follows:

$$TTR_{mr'} = ((f \times TTR_{mr}) + ((1 - f) \times TTR_{estimate}))$$

TTR_{mr} reflects the fastest change at the source, while $TTR_{estimate}$ tries to reflect the most recent change. Clearly $TTR_{mr'}$ takes into consideration both of them with the factor $f, 0 \leq f \leq 1$. The factor determines the weight given to recent changes and fastest changes.

The above algorithm has been shown [2] to provide very good fidelity and incur low network overheads and hence we use it in our algorithm.

3 CBA: The Cr Balancing Algorithm

The *Cr Balancing Algorithm (CBA)* continuously changes the *cr* associated with each of the data item constituting the portfolio based on the dynamics of the data, as well as the relative performance of the data item with respect to the other items in the portfolio. If false positives and false negatives are not to occur then the proxy must present a logically consistent view of the data items cached.

Mutual consistency in the value domain [2] is defined as follows. Cached versions of objects a and b are said to be mutually consistent in the value domain if some function of their values at the proxy and server are bound by δ. That is,

$$\forall |f(S_t^a, S_t^b) - f(P_t^a, P_t^b)| < \delta$$

Depending on the nature of the function f, the tolerance δ can be partitioned into two parts δ_a and δ_b such that $\delta_a + \delta_b = \delta$ and consistency of each individual object can be ensured by using the *Adaptive TTR Approach*.

CBA achieves this while aiming to reduce the network overhead – by pulling very rarely when there is very little chance of the threshold being exceeded. The CBA uses the *Adaptive TTR Algorithm* to compute the time after which a data item should be polled so that the coherency associated with that data item is not violated. How this is achieved is explained in the rest of this section.

3.1 Initial allocation of crs to stocks

Initially the cr of each constituent of the portfolio is calculated as follows (here the constituent data items are stocks). Let

$$v_i = n_i \times p_i$$

$$c_1 = \frac{v_2 + v_3 + \ldots + v_n}{v_1 + v_2 + \ldots + v_n} \times \frac{x}{n_1}$$

$$c_2 = \frac{v_1 + v_3 + \ldots + v_n}{v_1 + v_2 + \ldots + v_n} \times \frac{x}{n_2}$$

where
c_i is the cr of the i^{th} data item

$n_1, n_2 \ldots n_n$ is the number of stocks of each of the "n" stocks.
p_i is the initial price of the i^{th} stock
v_i is the value of the i^{th} stock
x is the *cr* of the portfolio (e.g., the threshold $100).

The formula for the *cr* is madeup of two factors. The first factor i.e.

$$\frac{v_2 + v_3 + \ldots + v_n}{v_1 + v_2 + \ldots + v_n}$$

takes into consideration the contribution of the stock to the overall portfolio. If the stock has a large contribution then that stock price should be maintained more accurately at the proxy. For this to happen the stock should be polled more often. Recall from the section on *Adaptive TTR Algorithm* that the $TTR_{estimate}$ is directly proportional to the *cr*. The smaller the *cr*, the smaller will be the $TTR_{estimate}$ and hence the smaller will be the TTR calculated. Hence the contribution of the stock to the portfolio is inversely proportional to the *cr*. If the contribution of the stock is more then the numerator of the first factor will be less and hence the *cr* will be less. Hence the first factor of the *cr* is inversely proportional to the contribution of the stock to the entire portfolio.

Multiplying the first factor by x apportions the *cr* of the portfolio for the individual stock. Since this applies to a total of n_i stocks, it is normalized by the number of stocks, giving the *cr* to be achieved by each stock.

	Company A Stock	Company B Stock
Let us consider an example:	Number of stocks n1 = 100	Number of stocks n2 = 200
	Price of the stock p1 = 10 $	Price of the stock p2 = 20 $

Using the above formula the *cr* is calculated as follows:

$c_1 = \frac{20\times200}{10\times100+20\times200} \times \frac{100}{100} = 0.8$

$c_2 = \frac{10\times100}{10\times100+20\times200} \times \frac{100}{200} = 0.1$

Company B has a greater effect on the portfolio and hence the *cr* of Company B stock is smaller. In effect, Company B stock will be polled more often than the Company A stock. This initial *cr* allocated to each of the data items (stocks in this case) should be changed dynamically depending on the change that occurs in the stock price. How CBA tries to achieve this is discussed next.

3.2 Dynamic adjustment of *cr*s of stocks

In the above example, according to the initial *cr* allocation, from a total of 100$ *cr* for the portfolio, Company A stock is allocated 80$ and Company B stock is given 10$ (100×0.1). Now if any of the stocks achieves the threshold allocated to it then the *cr* of the stock should be changed to reflect this. The main idea behind the CBA is to poll more often when the threshold is more likely to be exceeded and to reduce the polling frequency when the opposite is the case. When a stock achieves the threshold allocated to it, it should be polled more often.

In the above example, if the price of Company B changes by 0.1$ then the *cr* of the portfolio reduces to 90$. As the Company B stock has attained its threshold it should be assigned a new *cr*. Company B stock is changing such that it is taking the portfolio towards meeting its coherency requirement, hence it should be polled more often. Therefore the new *cr* assigned to the Company B stock should be less than the earlier one. The fact that the *cr* of the portfolio is reduced should also be reflected in the new threshold assigned to Company A stock. To achieve all this, the *cr*s of all the stocks are re-evaluated using the same formula as the one used in the initial calculation of the *cr*s, but taking the new portfolio threshold into consideration. As the *cr* of the portfolio has decreased, the *cr* of both the stocks gets reduced, thereby they are polled more often.

If the change in the stock price is such that it is taking the portfolio away from the threshold then the stock should be polled less often. Hence the cr of the stock is increased. If the change in the stock price is less than the cr allocated to the stock then the following actions are taken:

- If the threshold was crossed within δ time interval before the current time, then the cr is reduced by a small factor ϵ $(0 \leq \epsilon \leq 1)$:

  $$cr = cr \times \epsilon$$

  . Here, as in the *Adaptive TTR algorithm* we assume that the changes in the stock price in the recent past will be reflective of the changes in the near future. Applied to portfolio queries this implies that if the portfolio value changed more than the threshold within δ time interval of the current time then it is highly likely that this will recur in the near future.

- If the change did not exceed the threshold within δ time units of the current time then the chances of this happening soon are less. Hence the cr of the stock in question is increased by a factor γ $(\gamma \geq 1)$:

  $$cr = cr \times \gamma$$

  . Consequently if the portfolio value does not exceed the threshold for a long enough time then the cr will gradually increase to cr_{max}.

The factors δ , ϵ and γ can be changed and it will have an immediate bearing on the network overhead. We show the effect of changing these parameters in the next section.

The cr is not allowed to move outside a static window defined by two parameters - cr_{max} and cr_{min}. These bounds ensure that the cr computed by the CBA is neither too large neither too small - values that fall outside these bounds are set to $cr = max(cr_{min}, min(cr_{max}, cr))$. The value of cr_{min} and cr_{max} depend on the stock trace being considered. cr_{max} should be set such that it should be smaller than the maximum change seen in the stock trace. Both cr_{min} and cr_{max} are defined statically.

If the portfolio value change exceeds the threshold after a long time then the cr will have reached the cr_{max} value. Such a situation may arise if some stock price changes suddenly after being constant for a long time. Once the portfolio value change exceeds the threshold, according to the principle stated earlier, there is a high probability that it will remain so in the near future. To verify this the data item that has changed must be polled more often. Hence the cr of the stock in question is reduced by a factor β $(0 \leq \beta \leq 1)$:

$$cr = cr \times \beta$$

. In the next section, we evaluate the above technique to determine how well it meets the stated goals of threshold based portfolio tracking.

4 Experimental Evaluation

In this section we present the results of experiments conducted to evaluate the efficiency of our approach. We first present our experimental methodology and then our experimental results.

4.1 Experimental methodology

4.1.1 Simulation environment

The algorithm was evaluated using a prototype server/proxy that employed trace replay. Our experiments assume that the proxy has an infinitely large cache to store objects and that the network latency in polling and fetching objects from the server is fixed (this is because we are primarily interested in comparing network overheads based on the number of messages transmitted). We assume that the values of cr_{min} and cr_{max} are specified by the user. The proxy can keep track of the maximum change in the stock price seen so far and that can also be used for cr_{max}.

4.1.2 Traces used

The performance of the algorithm was evaluated using real-world traces. The presented results are based on stock price traces (i.e. history of stock prices) of a few companies 1 obtained from http://finance.yahoo.com. The traces were collected at a rate of 2-3 stock traces per second. Since the rate of change of any stock quote is much greater than even one change per second, the traces can be considered to be real time traces. All the experiments were done on the local intranet. The various data items of a portfolio can be obtained from different data sources. Hence no synchronization was assumed between accesses to the different data sources.

Table 1: Traces used for the Experiment

Company	Date	Time
Dell	Jun 1, 2000	21:56-22:53 IST
Intel	Jun 2, 2000	22:14-01:42 IST
Cisco	Jun 6, 2000	18:48-22:20 IST
Oracle	Jun 7, 2000	00:01-01:59 IST
Veritas	Jun 8, 2000	21:20-23:48 IST
Microsoft	Jun 8, 2000	21:02-23:48 IST

4.1.3 Metrics

The algorithm was evaluated using the following metrics (i) number of polls (normalized by the length of the trace) (ii) Fidelity of the portfolio. Fidelity can be measured based on the total time duration for which the proxy was oblivious of the portfolio value change exceeding the threshold at the server.

$$f = 1 - \frac{Total\ out\ of\ sync\ time}{Total\ trace\ duration}$$

where the Total out of sync time is the time duration for which the proxy was oblivious of the fact that the portfolio had attained its goal at the server i.e., the time duration for which there were *false negatives*.

The efficiency of the algorithm can also be measured in terms of the number of false positives and false negatives experienced. The algorithm should strive to keep the number of false negatives to a minimum, though a few false positives can be tolerated. The number of messages is expressed in terms of the number of pulls required per hundred entries in the source trace.

4.2 Experimental results

We evaluated the CBA using the traces. To do so, (unless otherwise stated) we configured the algorithm using the following parameters:

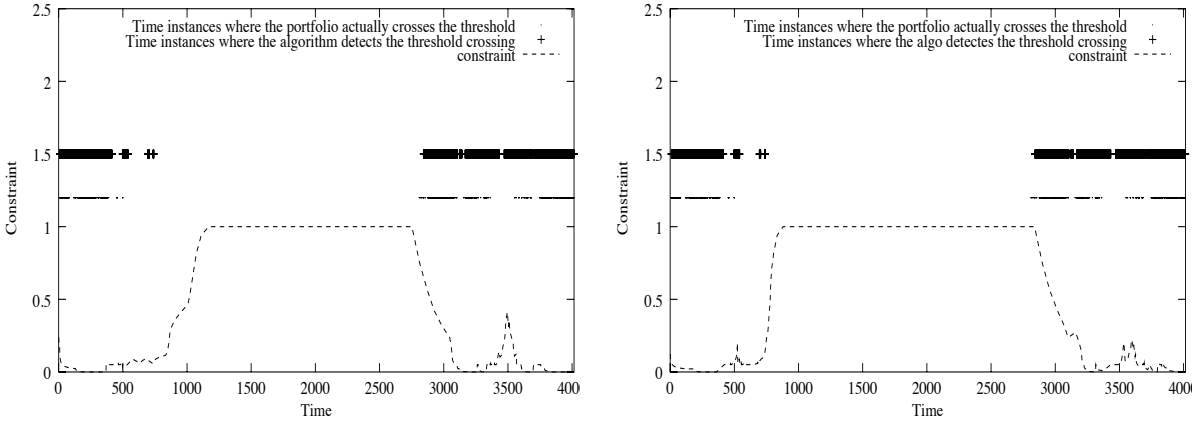
Symbol	Meaning	Value
δ	The time in seconds that governs the change in cr value	60 seconds
γ	Factor by which the cr is increased if (a) The change in stock price is less than its cr and (b) the value of the portfolio did not exceed its threshold within δ time	1.15
ϵ	Factor by which the cr is reduced if (a) The change in the stock price is less than its cr and (b) the value of the portfolio exceeded its threshold within δ time	0.85
β	Factor by which the cr is reduced if the portfolio is satisfied after a long time	0.75
cr_{min}	The minimum value of cr	0.05$
cr_{max}	The maximum value of cr	1$

The results were compared with an approach in which the cr was calculated once initially and was kept unchanged irrespective of the relative changes in the value of the data item.

We begin considering a portfolio comprising stocks of two companies, with 200 stocks of the first and 300 of the second, with the total value being:

$$200 \times price\ of\ stock1 + 300 \times price\ of\ stock2$$

We also experimented with portfolios with more than two stocks and the observations made for portfolios with two stocks are valid for all the experimented portfolios. Figure 2 shows the variation of the cr with time for the two stocks. Also shown in the two figures are the points in time when the portfolio (a) actually crossed the threshold (dots) and (b) the points in time when the portfolio crossed the threshold as per the CBA algorithm (For these two curves there is no significance to Y-axis).



(a) For stock 1 (b) For stock 2

Figure 2: Portfolio Changes and Variation of cr with Time

Figure 2 shows that in this trace the portfolio is satisfied at the beginning and towards the end of the observed time interval. In the middle part i.e. from time = 600 to time = 2700 the portfolio is not satisfied. Hence the

crs of both the stocks gradually reaches the cr_{max} value of 1\$. As the cr reaches the cr_{max} value the polling frequency is reduced and this helps is saving a lot of network overhead. Around time = 2700, when the portfolio is satisfied after a long duration the cr is reduced by the factor β (0.75). Thereafter as the portfolio remains satisfied the cr is decreased further. As cr decreases the polling frequency increases and this continues till the portfolio value change exceeds the threshold. If the value change is less than the threshold for δ time duration then cr will again increase till it reaches the cr_{max} value. Thus the CBA tries to reduce the network overhead by reducing the polling frequency when there is a lower chance of the change exceeding the threshold, but at the same time it does not compromise the fidelity by increasing the polling frequency once it detects that the change has exceeded the threshold.

In the experiments, the CBA detected that the portfolio was satisfied for about 1159 seconds when the portfolio was actually satisfied for 1016 seconds. This implies that there were a few false positives but they are not as harmful as false negatives. There were a further 101 false positives when the cr is not adjusted dynamically. Thus there were a lot more false positives in as compared to CBA. The network overhead of the latter was about 971 messages whereas the number of messages for CBA were only 551, a gain of about 43% in the network efficiency.

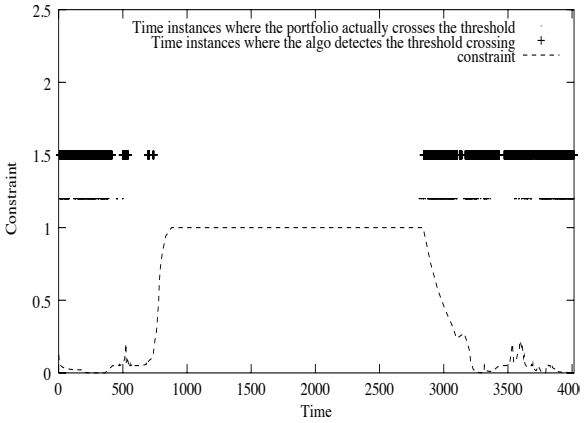
4.2.1 Effect of varying the parameters

Our technique provides several tunable parameters, namely, δ, β, γ and ϵ that can be used to control the algorithms behavior. In this section we illustrate the effects of varying these parameters on the fidelity offered by the algorithm and on the network overhead.

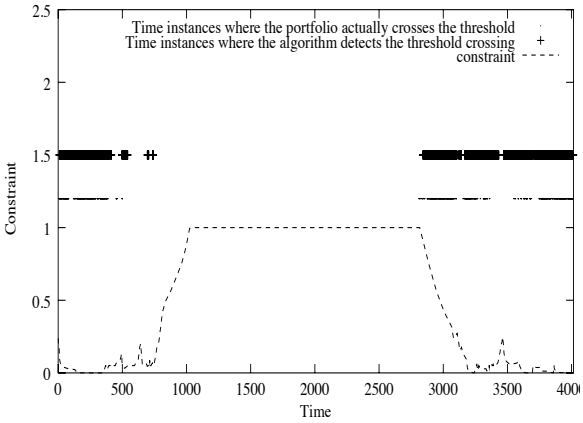
The factor δ governs the amount of elapsed time, having a bearing on the cr. If the change in the stock price is less than its cr, and if the portfolio had achieved its threshold within δ time, then the cr is reduced by the factor ϵ. If we increase the factor δ then a larger amount of history will influence the cr and it will be reduced by ϵ for a greater duration. Conversely if the value of δ is reduced, then the cr will not decrease and the network overhead will decrease. In this case as the cr is reduced by ϵ less frequently, chances are that the algorithm will have more false positives due to infrequent polling. The figure 3 shows the comparison of the variation of the cr for two different values of the δ. The effect of decrease in ϵ is very similar to that of increase in the value of δ i.e., with the decrease in the value of ϵ the fidelity offered by the algorithm will increase at the expense of message overhead. If the portfolio is satisfied after a long duration the cr is decreased by the factor β. The portfolio can be satisfied by either (a) a sudden momentary change in the stock price of any of the stocks or (b) a change in the stock price that persists for a long duration. If the polling is infrequent then the first case will lead to false positives. Hence to avoid this, the factor β helps. If the value of β is reduced then the cr will reduce by a larger amount and the chances of false positives will reduce. But if the change in the stock price persists for a long duration then an increase in the value of β will lead to unnecessary polling resulting in an increase in the network overhead. The figure 4 shows the effect of decreasing the value of β. A comparison of the two graphs reveals that in the second case (with smaller β) when the portfolio threshold is exceeded there is a sharp decrease in the cr. It was observed that in the second case when the β was set to 0.65 the number of false positives decreased sharply.

4.2.2 Comparison of network overhead

Figure 5 shows the comparison of the network overhead incurred by CBA and that when cr is not adjusted dynamically. The network overhead has been normalized with respect to the total length of the trace. The graph shows that the CBA has around 40% less network overhead. The graphs further reveal that as the threshold value

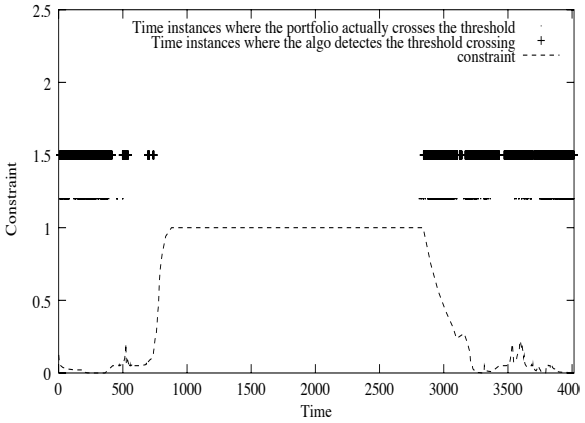


(a) Variation of the cr with $\delta = 120$

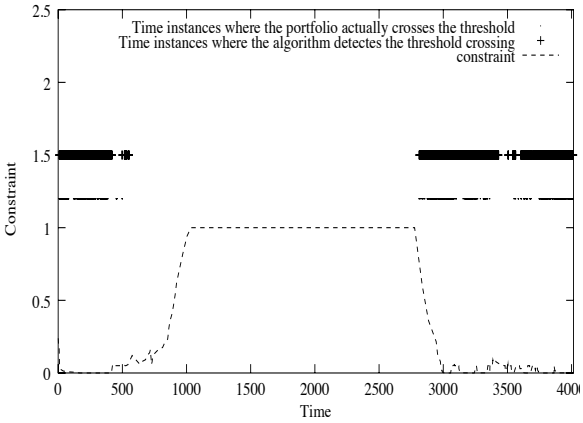


(b) Variation of the cr with $\delta = 60$

Figure 3: Effect of change in δ on cr



(a) Variation of the cr with $\beta = 0.75$



(b) Variation of the cr with $\beta = 0.65$

Figure 4: Effect of variation of β on cr

of the portfolio is increased the network overhead goes on decreasing. As the threshold of the portfolio increases the number of times that the portfolio crosses the threshold decreases. In the CBA the number of pulls increase only when the threshold is reached. For the rest of the duration the CBA increases the cr so that the number of pulls is reduced. Hence as the threshold increases the pulls i.e the network overhead decreases due to reduced number of threshold crossings. In case where the crs are allocated statically , the cr allocated to each of the data item increases with the increase in the threshold. With an increase in the cr the value of TTR calculated by the *Adaptive TTR Algorithm* increases and hence the network overhead reduces to some extent. Thus by considering the dynamics of the data the CBA succeeds in reducing the network overhead compared to the case when the cr is statically allocated.

5 Concluding Remarks

In this paper, we presented a new technique for monitoring the stock quotes in the case where the user is interested in a group of stocks. Our algorithm is entirely proxy based and it assumes no push support from the server[8]. The

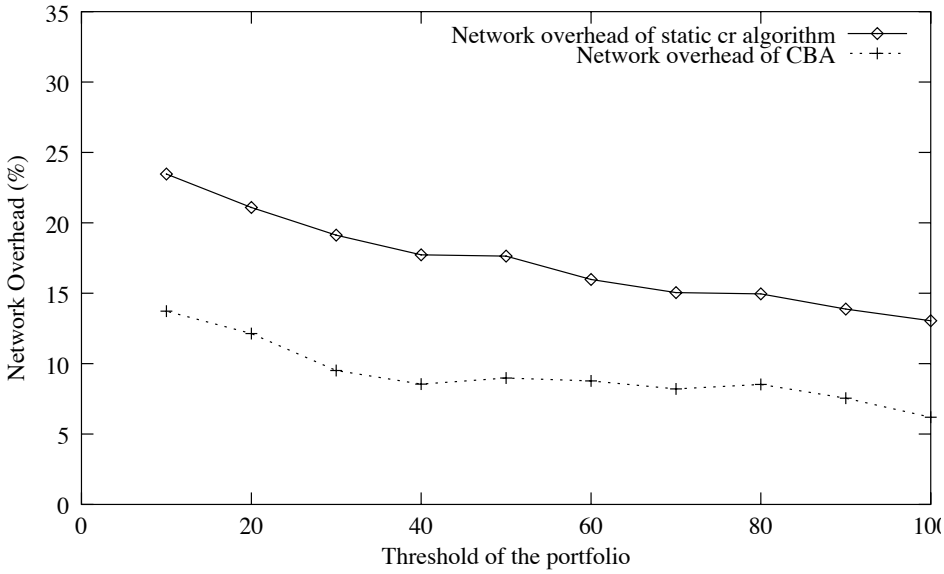


Figure 5: Comparison of the network overhead of CBA and pure Pull

algorithm requires some amount of history to be kept, which can be easily supported by the proxy, as the proxy caters to less number of clients vis-a-vis the server. A useful feature of our technique is that it has several adjustable parameters which can be used to tune to algorithm to get the desired fidelity and network overhead characteristics. The network overhead offered by the CBA is around 40% less than that offered by the pure pull approach. The CBA achieves this objective by considering the stocks in a portfolio as a semantic unit, in the sense that it reduces the polling frequency of all the stock of the portfolio if the stock prices are changing such that there is very little chance of the portfolio being satisfied.

Portfolio queries can be considered as type of continuous queries [12, 13]. These queries can also be implemented by using other techniques such as PAP [5] and leases[10, 11]. This will require push support at the server and is left as future work.

References

[1] J.C.Mogul Squeezing More Bits Out Of HTTP Caches *IEEE Network Magazine, 14(3):6-14, May 2000.*

[2] R. Srinivasan, C. Liang and K. Ramamritham, Maintaining Temporal Coherency of Virtual Data Warehouses, *The 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, 1998.*

[3] J. Gwertzman and M. Seltzer, World-Wide Web Cache Consistency., *In Proceedings of 1996 USENIX Technical Conference, January 1996.*

[4] B. Urgaonkar, A. Ninan, M. Raunak, P. Shenoy and K. Ramamritham, Maintaining Mutual Consistency for Cached Web Objects *International Conference on Distributed Computing Systems , April 2001.*

[5] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy, Adaptive Push-Pull: Dissemination of Dynamic Web Data *10th International World Wide Web Conference , Hong Kong, May 2001.*

[6] P. Cao and S. Irani, Cost-Aware WWW Proxy Caching Algorithms., *Proceedings of the USENIX Symposium on Internet Technologies and Systems, December 1997.*

[7] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz and K. J. Worrel A Hierarchical Internet Object Cache., *Proceedings of the 1996 USENIX Technical Conference, January 1996.*

[8] A. Iyengar and J. Challenger, Improving Web Server Performance by Caching Dynamic Data., *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USEITS), December 1997*.

[9] W.R. Stevens, *TCP/IP Illustrated Volume 1* Addison Wesley, 1994

[10] V. Duvvuri, P. Shenoy and R. Tewari, *Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web. InfoCom March 2000*.

[11] J. Yin, L. Alvisi, M. Dahlin and C. Lin, Hierarchical Cache consistency in a WAN., *Proceedings of the USENIX Symposium on Internet Technologies and Systems, October 1999*.

[12] J. Chen, D. Dewitt, F. Tian and Y. Wang, NiagraCQ: A Scalable Continuous Query System for Internet Databases, *ACM SIGMOD Conference on Management of Data, May 2000*.

[13] L. Liu, C. Pu and W. Tang, Continual Queries for Internet Scale Event-driven Information Delivery. *IEEE Trans. on Know. and Data Engg., July/August 1999*.