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

Manish Bhide, Krithi Ramamritham
Laboratory for Intelligent Internet Research
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, India 400076
manishb,krithi@cse.iitb.ac.in

Prashant Shenoy
University of Massachusetts
Amherst, MA 01003
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.

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 is not scalable. 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, with minimal loss of fidelity in the responses provided to the users.

1.1. Maintaining the 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 [2, 3] 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)* [8] and client polling [6] 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 port-

folio that exceed a certain threshold. We call this threshold the *coherency requirement* (cr). To illustrate, a user may have 50 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 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 and (b) the coherency requirement for the portfolio.

Let $S(t)$, $P(t)$ and $U(t)$ denote the value of an individual stock at the server, proxy cache and the user, respectively. Then, to maintain coherency we should have $|U(t) - S(t)| \leq tcr$ for each of the stocks, where tcr is the constraint allocated to each individual stock. Further in the realm of portfolio queries, whenever the stock prices at the source(s) are such that a user's portfolio value exceeds the specified threshold, the user should also be aware of this fact. That is, the following equation should 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.

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 change has 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. To quantify the misperceptions, we define *fidelity* of the portfolio to be 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. In addition to fidelity, another measure of evaluating the algorithm is to measure the number of false positives and false negatives that have occurred.

[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.

1.2. Challenges in using HTTP to Maintain Coherency of 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 bit in the HTTP header to track 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. In addition, if a portfolio consists of stock information served by different servers, then the push approach will require cooperation between these servers, a requirement that may not always be satisfiable. So, our goal is to work within the standard HTTP protocol and 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 individual data items would be to assign a small (i.e., stringent) 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 low and that of company B can be high. This will help in reducing the false negatives and it will also help in saving unnecessary pulls of the B's 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 briefly 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 maintain coherency of individual data items, a proxy computes a *Time To Refresh (TTR)* for the data item. 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. We use the *Adaptive TTR Algorithm*[9], 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.

This algorithm provides good fidelity and incurs low network [9] 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 items constituting the portfolio based on the dynamics of the data, as well as the relative contribution of the data item to the portfolio with respect to the other items.

Mutual consistency in the value domain [9] 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. 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

$$v_i = n_i \times p_i$$

$$c_1 = \frac{v_2 + v_3 + \ldots + v_N}{v_1 + v_2 + \ldots + v_N} \times \frac{x}{n_1 \times (N-1)}$$

$$c_2 = \frac{v_1 + v_3 + \ldots + v_N}{v_1 + v_2 + \ldots + v_N} \times \frac{x}{n_2 \times (N-1)}$$

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. Smaller is the cr, smaller will be the TTR calculated by the *Adaptive TTR Algorithm*. Hence the contribution of the stock to the portfolio should be 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. As a result 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. The factor $N-1$ is needed to normalize the constraints so that the following equation is satisfied -

$$c_1 \times n_1 + c_2 \times n_2 + \ldots + c_N \times n_N = x$$

Let us consider an example:

Company A Stock	Company B Stock
Number of stocks n1 = 100	Number of stocks n2 = 200
Price of the stock p1 = \$10	Price of the stock p2 = \$20

The change to be tracked is of \$90. Using the above formula the cr is calculated as follows:

$c_1 = \frac{20 \times 200}{10 \times 100 + 20 \times 200} \times \frac{90}{100 \times (2-1)} = 0.72$

$c_2 = \frac{10 \times 100}{10 \times 100 + 20 \times 200} \times \frac{90}{200 \times (2-1)} = 0.09$

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 crs of stocks

In the above example, according to the initial cr allocation, from a total of \$90 cr for the portfolio, Company A stock is allocated \$72 and Company B stock is given \$18. 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 \$0.09 then the cr of the portfolio reduces to \$72. 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 crs of all the stocks are re-evaluated using the same formula as the one used in the initial calculation of the crs, 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 polling for the stock should be reduced. 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 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.

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}.

As detailed in [1], the performance of the algorithm was evaluated using real-world traces . 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 correct state of the portfolio.

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. We configured the algorithm using the parameters shown in Table 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. This approach is the simple TTR approach which simply uses the *Adaptive TTR Algorithm* to calculate the TTR based on the initial allocation of cr to the stocks.

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 1 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).

Figure 1 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 change in the value of the data item 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. These are a lot more 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.

Our technique provides several tunable parameters, namely, δ, β, γ and ϵ that can be used to control the algorithms behavior. In [1] we discuss the effects of varying these parameters on the fidelity offered by the algorithm and on the network overhead. In [1] we present results which show that 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.

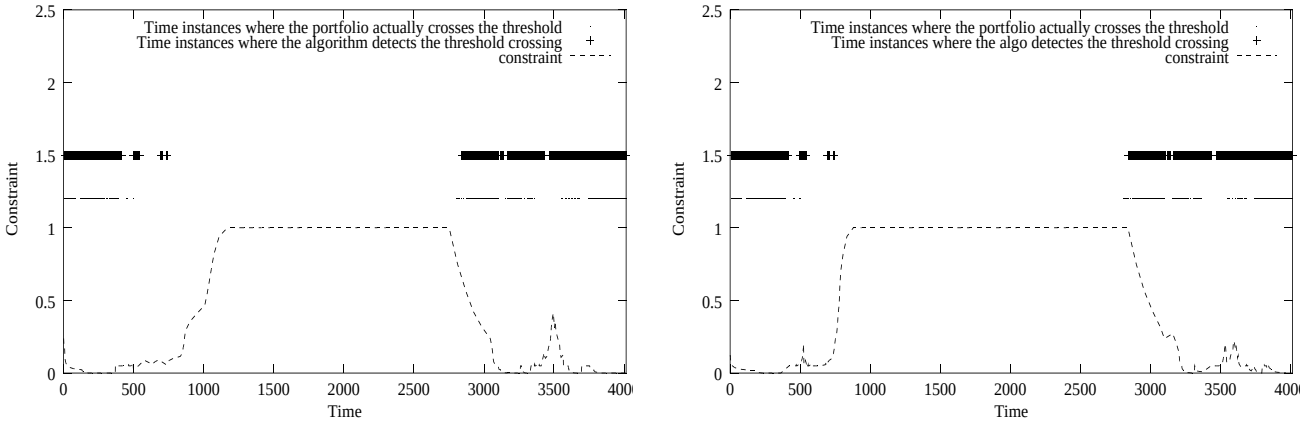
5. Concluding Remarks

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[7]. The 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 static cr 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.These queries can also be implemented by using other techniques such as PAP [4] and leases[5, 10]. This will require push support at the server and is left as future work.

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 exceeds the threshold after a long time	0.75
cr_{min}	The minimum value of cr	\$0.05
cr_{max}	The maximum value of cr	\$1

Table 1. Value of Parameters used in the experiments



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

Figure 1. Portfolio Changes and Variation of cr with Time

ACKNOWLEDGEMENTS

This research was supported in part by Tata Consultancy Services, Intel, IBM, and Microsoft, as well as NSF grants IRI-9619588, CDA-9502639, CCR-0098060, CCR-9984030, CDA-9502639, and EIA-0080119.

References

[1] M. Bhide, K. Ramamritham, and P. Shenoy, Efficiently Maintaining Stock Portfolios up-to-date on the Web, available at (www.cse.iitb.ac.in/ krithi/papers/portfolio.ps), March 2002.

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

[3] 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.*

[4] 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.*

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

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

[7] 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.*

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

[9] 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.*

[10] 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.*