Exploring the Benefits of a Continuous Consistency Model for Wireless Web Portals [*]

Jagadeeswaran Rajendiran, Jaidev Patwardhan, Vijay Abhijit, Rahul Lakhotia, and Amin Vahdat
Department of Computer Science
Duke University
{*jags,jaidev,abhijit,rahul,vahdat*}*@cs.duke.edu*

Abstract

Wireless devices currently provide real-time access to personalized information such as headlines, email, stock quotes, and online auctions. Retrieving all updates to such rapidly changing information is wasteful of both network bandwidth and battery power. Existing consistency models for such services allow for only coarse-grained timeouts on how often information should be retrieved. In this paper, we argue for the benefits of a continuous consistency model for user access to Internet portal services. Using this model, users are able to specify the maximum error in their view of the data. For instance, users may specify that they wish to receive an updated stock quote only if the users view diverges from the actual value by more than 3%. Services may also use application-specific semantics to control data consistency—e.g., new bids carry more weight as the auction draws to a close. We use a simulator to model the bandwidth and energy benefits available from a more flexible consistency model. Such benefits depend upon the rate at which underlying data values change. To capture representative distributions, we use a trace-based study of updates to weather, news, and stock quotes from a popular portal to determine representative distribution ranges. Our initial results indicate bandwidth and energy savings that increase as users are willing to tolerate larger bounds on data accuracy.

1. Introduction

Today, ubiquitous wireless devices promise real-time views of rapidly changing data customized to the interests of individual users. For example, wireless portal services can provide snapshots over news headlines, stock quotes, sports scores, weather forecasts, and a user's mail messages in real time. However, rapid asynchronous updates can be both distracting to the end user and wasteful of limited system resources, such as wireless bandwidth and battery power. One common approach to reducing resource consumption and asynchrony is to periodically pull updates from the server, with the user able to pull more frequently if needed. Of course, this often results in users who poll for new email messages or changes to stock quotes or sports scores when no changes actually take place to the underlying data.

We observe that the fundamental mismatch between the user's desire to have "up to the second" accurate information and the need to reduce system resource consumption lies in the limited ability to specify consistency requirements. For example, users currently specify the maximum staleness of their data in real time, but have no ability to specify the maximum amount that the underlying data has changed. Consider the example of stock quotes. Most portal services indicate that stock quotes are delayed by at least 15 minutes, whereas many users would be more satisfied with a guarantee that a quote is accurate within, for example, 3% of its actual value. In this paper, we explore the benefits of using a more flexible consistency model to control the accuracy of data depicted to end users. We use the example of users accessing customized information through a wireless portal service. However, we believe the results can be generalized to any group of distributed machines maintaining a shared view of rapidly changing data (e.g., cooperative web caching or content distribution networks).

In this context, this paper makes two principal contributions. First, we describe a simulation environment that models different update patterns for a generic set of target data. The simulator also tracks the consistency requirements of a population of users, pushing necessary updates to prevent violation of any consistency bounds. The system tracks bandwidth and energy characteristics on a per-user basis, allowing us to compare resource consumption at various consistency levels relative to a system that maintains strong consistency. Reduced resource consumption is important, but the system must provide sufficiently accurate

[*]This work is supported in part by the National Science Foundation (EIA-99772879,ITR-0082912,ITR-0082914). Vahdat is also supported by an NSF CAREER award (CCR-9984328). Additional information on the TACT project can be found at `http://www.cs.duke.edu/ari/issg/TACT/`.

information to be adopted by end users. While this accuracy is subjective (e.g., a day trader will not tolerate anything less than real-time quotes), we set out to determine potential distributions of updates to data of interest to Internet users. Thus, the second contribution of this work is a trace-based study to the rate and weight of updates to news headlines, weather forecasts, and stock quotes. While we do not argue that these initial measurements are representative, our ability to fit the geometric distribution to these updates allows us to study the benefits of our model for concrete points in the simulation space.

The rest of this paper is organized as follows. Section 2 provides background on the consistency model used for this work. Section 3 describes the trace methodology that is used to determine initial models for how data might change in a portal service. Section 4 describes the simulator we built to carry out our experiments. Section 5 presents a detailed description and analysis of our results. Section 6 presents our conclusions and future work.

2. Background

In this section, we discuss the consistency model used for our study. Today, consistency of web content is typically expressed in terms of real-time staleness. Thus, users receive guarantees that a stock quote is, for example, no more than 15 minutes old, or that a sports score will be refreshed every minute. While such strict age-based notions of consistency are useful, the passage of time does not always directly correlate with the rate at which the underlying data is changing. For instance, a stock may not change in value at all during a 15 minute value, or it may change dramatically in a much shorter time frame.

In our earlier work [13], we designed and evaluated TACT, a continuous consistency model that allows us to dynamically trade reduced consistency for improved performance and availability for replicated Internet services. This tradeoff can be continuously made allowing the service to adapt to changing client, network and service characteristics. TACT uses three metrics to flexibly bound the consistency of a replicated service relative to its peer replicas. *Numerical Error* specifies the maximum weight of updates not seen locally. *Order Error* is the maximum number of local updates that have not been propagated to remote replicas and thus have not established their final commit order. Finally, *Staleness* is the maximum amount of elapsed time before data is pushed to a replica. In the context of portal services, such as instant messaging, email, online auctions, and news, order error captures application sensitivity to the ordering of updates (e.g., in the case where multiple bids are being buffered at a single replica). Because such ordering requirements are subjective, we restrict our attention to the benefits of numerical error and staleness though we believe that order error is very useful for a range of distributed services as shown in [15].

In our model, users specify their desired numerical error and staleness in data delivered through a customized portal service. We hypothesize that allowing users to flexibly control both numerical error and staleness will prevent polling that results in unchanged or nearly unchanged data (wasting both bandwidth and energy). Thus, for example, users may specify that they wish to be notified of stock price changes larger than 3% with maximum staleness of 15 minutes. Consistency is specified at the granularity of "conits" [15] (consistency units), logically a set of related data items. Larger conits induce less overhead at the server, while smaller conits provide finer control over data accuracy.

In this paper, we investigate the use of a particular consistency model that allows applications to bound inconsistency. For brevity, we simply note that there are a number of related efforts in "variable consistency" [2, 3, 6, 8, 10], optimistic consistency [4, 5, 7, 11, 12], and more traditional database consistency models [1]. We believe that our approach of evaluating the benefits of a various consistency models for portal services applies to these related efforts.

3. Trace

3.1. Overview

To obtain a representative distribution for the data for our services, we collected trace data from three sample web services. The traces also serve to validate the results generated by our simulator. We used the following services: Yahoo! stocks, Yahoo! news, and Weather.com weather. These three services are extremely popular among the online community. They also exhibit diverse characteristics, being updated at varying intervals. The nature and quantity of the data provided by the sites also differ substantially. We employed the parameters of Numerical Error and Staleness to perform our processing of consistency. Modeling staleness is straightforward in all three cases as the data is updated periodically. For News, numerical error is difficult to model, but we settle on the number of new headlines generated as the measure.

3.2. Stock Trace

Stocks provide the most dynamic data content. Through real-time quotes, the stock values update virtually every second. We chose stock quotes for Microsoft, Sun Microsystems, Intel Corp, Oracle Corp, Cisco Systems, Dell Computer, Applied Materials, Infosys Tech, General Motors and PSIX as our sample set. The trace queried for values every

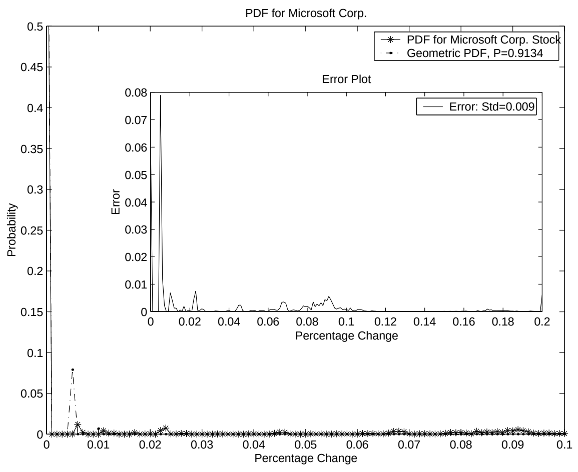

Figure 1. Fitting a Geometric distribution to stock value updates.

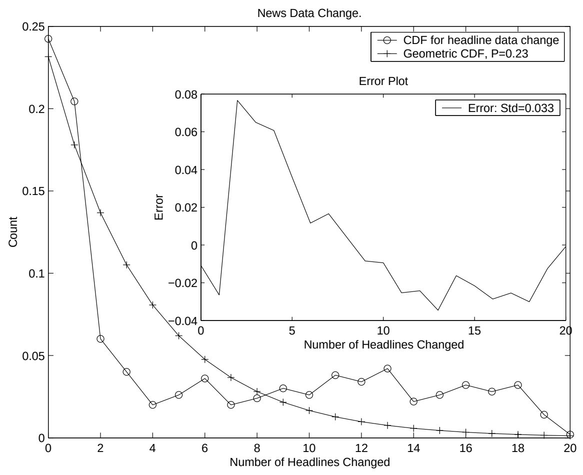

Figure 2. Fitting a geometric distribution to news updates.

three seconds and it was carried out between 10am to 4pm on five consecutive week days.

We measure the change between two readings and calculate the probability of change. We have tried to fit the data into several distributions. The main characteristic of the data is a very high probability for no change (around 0.91). The geometric distribution approximates the data to a high degree and can be used to model the data for our stock trace. Figure 1 shows a magnified view of the distribution for the Intel Corp. stock, and the geometric distribution that can be used to model it. The parameter for the fitting distribution is P=0.91. The parameter P of the geometric distribution gives the probability of zero occurrence. It also controls how fast the probability of occurrence drops as we move along the X-axis. The error in our fit had a standard deviation of 0.009. This means that our estimate deviates from the actual data by 0.009, implying a good fit.

3.3. News Trace

News headlines are another example of dynamic data. However, unlike stock quotes that vary on the granularity of seconds, variation in headlines are more infrequent and less predictable. We used Yahoo! News headlines to collect our data trace. We find that headlines are updated every forty minutes on average. We have collected data every twenty minutes from the site, over a period of two weeks in early November 2000.

Each headline is stored as a string in a file. To calculate the change, for every set of headlines we computed the number of new headlines, and used that number as the "change". The best-fit distribution once again is a geometric distribution. Figure 2 shows the distribution for News headlines and the geometric distribution that can be used to model it. This distribution used a parameter of P=0.23 and had a standard deviation on 0.033 from the data. As can be seen in the graph and from the standard deviation, the variation is very irregular, and it is difficult to get a good fit for it.

3.4. Weather Trace

Weather data is the third type of service that we consider. We collected weather data from ten cities, widely distributed geographically and climatically. The site data is updated approximately once an hour. In order to ensure that we did not miss updates, we collected readings at forty minute intervals. The data we collected included the following readings: temperature, wind speed, wind chill factor, humidity, and dew point. We find that the geometric distribution with parameter P=0.67 can best model the data, and results in a standard deviation of 0.021 from the actual data. This value of standard deviation implies a reasonably good fit. Due to the irregularity in the data, it is difficult to do much better than this. Figure 3 captures the results of this analysis for the city of Tokyo.

4. Simulation

In the previous section, we described trace results of update distributions to a number of popular Internet services. In this section, we describe our simulation environment used to quantify the bandwidth and energy savings available from using a flexible consistency model to bound the error in a client's view of data. Since the magnitude of such benefits depends upon the rate of change of the underlying

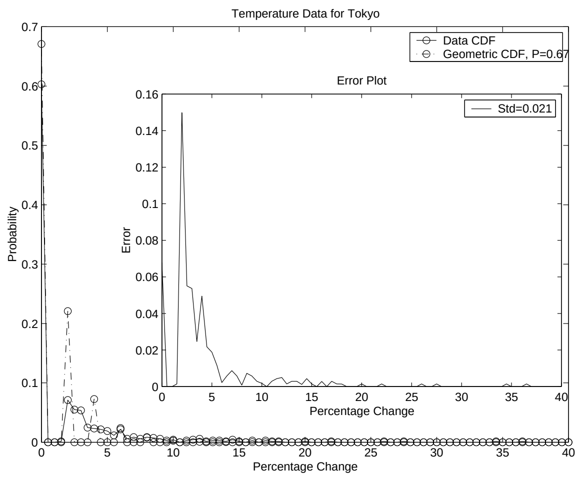

Figure 3. Fitting a geometric distribution to weather updates.

data, we use results from our trace-driven study to drive our simulations.

4.1. Simulator Overview

This section describes our simulator. The simulator is implemented in C++ to mimic the operation of the web services described in section 3. We model the services using the geometric distribution and variable update arrival rate. Users specify their desired consistency level using staleness and numerical error. Staleness specifies the longest time the client is willing to go without an update. Numerical error quantifies the maximum percentage error that clients are willing to tolerate before receiving an update. For instance, a numerical error of 0.05 for stock quotes specifies that clients will be updated whenever their local view of the stock price diverges by more than 5% from the “real” stock price. The client is free to set any interval to “pull” the data. By pull, we refer to explicitly requesting a server update. A “push” is the complement of a “pull”. During a “push” the server detects a consistency violation and transmits the data to the client. Thus, clients set a consistency bound at the server, but are free to get the data at any time.

The simulator implements a flexible bi-directional anti-entropy mechanism [11]. To test the performance of the simulator over the entire range of numerical error and staleness values, each (numerical Error, staleness) tuple value corresponds to a single client. The simulator does not model contention at the server or any network congestion. Furthermore, overheads associated with TCP connection establishment/tear-down are not considered and bandwidth consumed by protocol headers are ignored. In general, these simplifying assumptions tend to understate the benefits of our approach. Our simulator assumes the use of delta encoding [9] for data transfers. Thus, instead of retransmitting the entire HTML page, our system transfers only the changed data.

4.2. Simulation Parameters

The bandwidth savings essentially correspond to savings in the amount of data transferred over the network at different consistency values relative to strong consistency. To compute these savings, we add the savings accrued from pushes and pulls. In performing these calculations, we use a slightly higher overhead for pull, over the penalty chosen for push. Since pull involves an additional level of interaction in terms of requests and acknowledgments between the server and the client, this is reasonable. The transfers were assumed over an ideal 19.2kbps wireless link. We model stock and weather services a little differently from the news service. The reason for this is that while changes in news involve new data being generated in addition to existing data, changes to stocks and weather imply a change in the value of an existing data item. On the other hand, the various percentage changes for headlines point to a different number of headlines being generated. Hence, we choose a 5 byte value for any change in the values of stocks, 100 bytes for headlines and 50 bytes for a change in weather values (These are the values used with delta encoding).

Energy calculations present certain additional difficulties since idle energy is significantly different from the energy required to send and receive. The specifications we choose are taken from a mobile phone data-sheet, with power parameters 0.05W for Idle, and 0.6W to either send or receive. Our energy calculations are entirely restricted to the Network Interface and do not take into account other system devices. We assume that the device switches instantaneously from the idle to the send/receive mode. If no transfers are taking place, we assume that it instantly goes to low power mode. We calculate the energy consumption for receiving updates and add this value to the energy consumed while in idle mode.

4.3. Simulator Operation

The simulator tries to establish anti-entropy between each service and all the clients subscribed to it according to the client-specified numerical error and staleness values.

Each service is modeled by a distribution that maps the probability of updates to the percentage change in data. The simulator analyzes the state of each service every second. For each service i, a biased coin is tossed to decide the weight of an update based on a specified distribution function. For each update, only the difference in the data maintained at the server and the client is transmitted, as obtained through the use of delta encoding [9]. The simulator can be in one of three modes for each client at every update: quiescent, push, pull. If the current update does not cause a push/pull, then the update is accumulated but is not propagated to the client. If the accumulated updates exceed the numerical error or staleness specified by the client, then the service pushes the data to the client.

The simulator operates on a time granularity of one second. We model services such that an update is expected once every X seconds (where X is service dependent). We set the amount of data generated to different amounts for each of the services depending on the value returned by our distribution and the service. The network interface is assumed to be a wireless device. We can modify the characteristics of the wireless connection through a configuration file. These include connection speed and various power consumption levels.

4.4. Simulator Verification

In order to verify the results generated by our simulator, we adopt a two-fold strategy. We modify the simulator to dump the entire intermediate push/pull behavior of several different clients and manually track these parameters for possible bugs in our code. We repeat this test over a number of simulations and this leads us to the next step of verification.

The second technique of verification involves writing a small "verifier" in C, from scratch. This program tries to replicate the calculations that we use in our simulator model but without many aspects of the simulator, which leads to great simplification. We consider only one client subscribed to all three services, and use the same consistency bounds for all three services. We make modifications to the simulator to give us the random data generated as well as the corresponding data changes. We then feed this data to the verifier along with the the numerical error and consistency values of one client whose performance was tracked in the simulator. We compare the values generated by the verifier, vis-a-vis the parameters of bandwidth and energy with the corresponding output from the simulator. We find that the results were identical over all combinations of numerical error and staleness used for verification.

We further validate the simulator by using data from our traces as input. This helps us verify that it does indeed produce a distribution that matches our input data. We compare the output generated using the trace data with that produced when the simulator generates its own data. We use the root mean square error as a measure of how well our simulator generates data to match the underlying traces. We vary the parameter P and calculate the error.

Table 1 below shows the root mean square errors (percentage values) as obtained for various values of p, ranging

p	rms-error	p	rms-error
0.1	24.0%	0.6	11.2%
0.2	18.6%	0.7	8.34%
0.3	17.4%	0.8	5.20%
0.4	15.7%	0.9	2.65%
0.5	13.6%	0.9	12.57%

Table 1. RMS Error for artificial data with respect to real stock traces

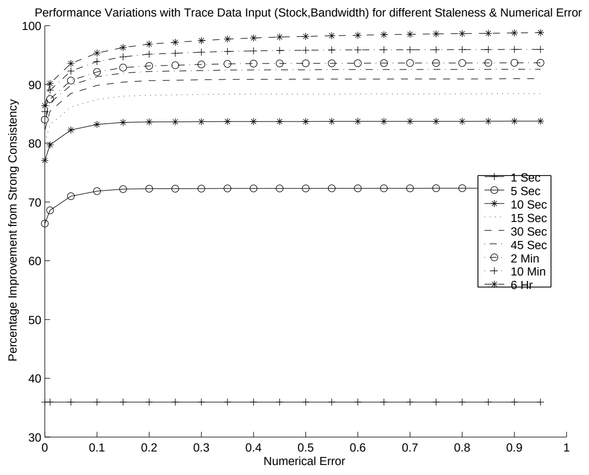

Figure 4. Performance with trace input to simulator (Stocks).

from 0.1 to 0.91 for stock quotes.

An rms error of nearly 24% signifies a rather poor fit between the trace data and output produced by the simulator with a 0.1 p value. We find that the lowest error is obtained at p=0.91, the same p value that gives us the best fit with the trace data. We think that the corresponding error value of 2.58% is a reasonably good fit, given the randomness of the input data. Figure 4 shows the performance of our system using trace data.

5. Results

We run simulations using the geometric distribution for three different services. Each service has different characteristics that closely match real services like Weather, Headlines and Stocks at Yahoo!. Each simulation examines the services based on two parameters, bandwidth and energy.

The values of numerical error and staleness are provided as input parameters to the simulator. The simulator is configured to vary staleness for a given value of numerical error unless otherwise stated. The simulations run for a total of 1008 clients, each corresponding to a unique value of staleness and numeric error. The pull-time for each client is set to 3000 seconds. To remove any random effects, we run the simulations fifty times and the data from the fifty readings is averaged to give the final results.

The main goal of our evaluation is to show that there are improvements available from using variable consistency. Thus we measure improvements with respect to performance at strong consistency.

5.1. Bandwidth

In general, we observe large improvements in bandwidth for all three services implemented. Each service has a different data unit size. Stocks have the lowest with 5 bytes, weather 50 bytes, and headlines have a size of 100 bytes per headline. Because less data must be pushed with reduced consistency, the amount of bandwidth left for other applications is higher.

Bandwidth improvements of up to 95% are noted for lower consistency settings. Figures 5, 6 and 7 show the bandwidth savings as a function of consistency for sample update distributions corresponding to our traces for weather, stocks, and headlines (geometric distribution). In each case, the graph plots savings in bandwidth relative to strong consistency as a function of numerical error. Different curves correspond to different staleness values. A numerical error value of 0.1 implies that stock values will be pushed if they experience changes in value larger than 10%. It is interesting to note that, the curves rise sharply with much of the bandwidth savings taking place for relatively low values for numerical error (with the knee of the curve at between 5-10% for the three graphs). We conclude that for our target update distributions, users can obtain bandwidth savings while also receiving strong guarantees regarding the maximum "inconsistency" in their target data.

The graphs clearly show that different values of numerical error and staleness give different levels of performance. For stocks, numerical error has a very small effect. This is because of the fact that stock updates are rather small. For most updates, the change is only around 0.1%. Hence, numerical error has a much smaller effect than staleness. The data is typically pushed because of staleness before the numerical error is violated. The effect of numerical error for other services on performance is strong at first, and then drops sharply. Staleness also exhibits similar behavior with initial increases in staleness causing a sharp increase in performance. At higher values, the effect dips. One also notices that each curve has a different starting point, pointing to the fact that the base performance increase also goes up as we lower the bounds on staleness. The results start flattening out as staleness starts dominating over numerical error. The graphs depend on other factors like data generation rate, maximum amount of data generated, distribution pa-

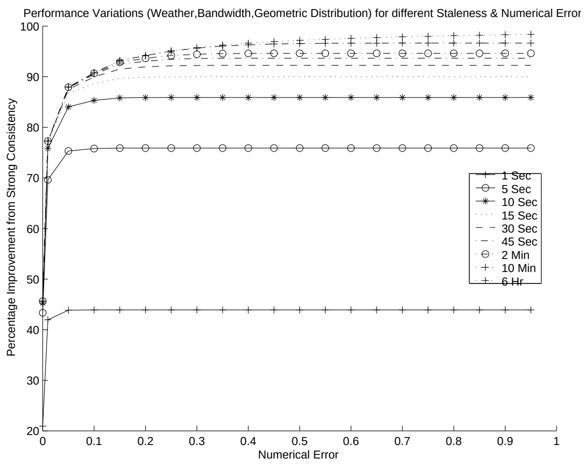

Figure 5. Bandwidth savings as a function of numerical error and staleness (Weather, Geometric distribution).

rameters and pull time. Some of these effects are presented below.

5.2. Energy

Improvements in the amount of energy consumed are lower than the other parameters for all three services. This is largely due to the fact that the clients spend most of their time in an idle state and a little variation from that does not lead to a very large reduction in energy usage. However, we do get an improvement for all lower consistency settings, varying from a 6% to 20% reduction in consumed energy. One representative result is depicted in Figure 8. One reason why the energy savings are modest relative to bandwidth is that only a small amount of data is transferred for each update (recall that we assume delta encoding for transmitting updates). Thus, we believe that energy savings will accrue with a larger number of tracked data items. For example, a typical portal user may track dozens of stocks and news headlines simultaneously. Updates to individual stock quotes/news headlines would incur overhead independently. In the simulator described so far, we track only the value of a single stock. Further, we do not model any of the bandwidth or energy overheads associated with TCP connections, TCP/IP packet headers, or HTTP protocol overhead. We measure performance of a client tracking multiple data items using a modified version of our simulator in the next subsection.

5.3. Multiple Data Items

So far, we have limited our experiments to only one item per service - that is, we have monitored only one stock, one

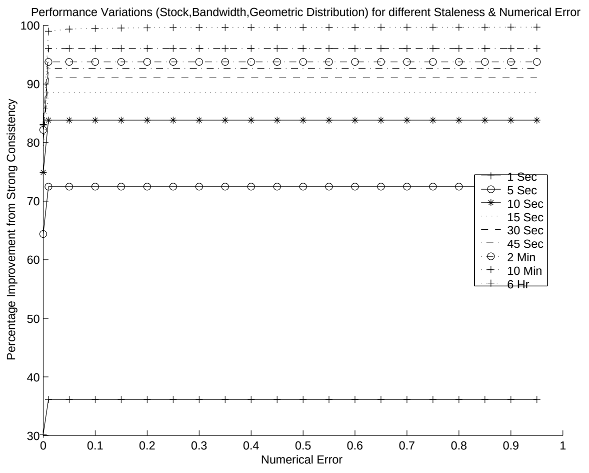

Figure 6. Bandwidth savings as a function of numerical error and staleness (Stock, Geometric distribution).

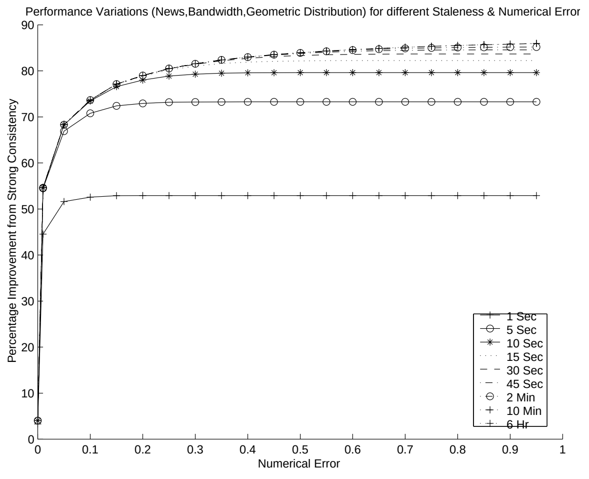

Figure 7. Bandwidth savings as a function of numerical error and staleness (News, Geometric distribution).

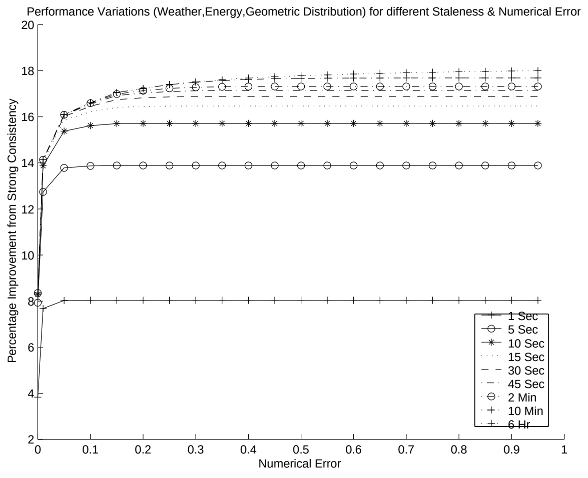

Figure 8. Energy savings from various consistency states (Weather, Geometric distribution).

headline and one piece of weather data. In order to explore the benefits of our scheme further, we extend our model to capture the savings accrued from tracking multiple instances for each service.

For our experiments, we include thirty stocks, thirty headline items and thirty pieces of weather data and run our simulation for the same time period as before. We find that while the percentage savings with respect to the strongest consistency client remains nearly the same with one item per service as well as multiple ones, the savings logged in terms of raw numbers of bandwidth and energy increase almost linearly with the number of items monitored. We argue that with multiple items per service, a percentage improvement in the range of 80 or 90 with respect to strong consistency would translate into considerable savings in the amount of data saved from being transmitted (high bandwidth savings) as well as a substantial increase in the lifetimes of the batteries of mobile devices, resulting from savings in the amount of energy expended. For instance, while the maximum saving logged with monitoring a single stock item are nearly $3.6 * 10^5$ bytes over a 25,000 second simulation period, the corresponding values for 30 stock items are $10.2 * 10^6$ bytes, both based upon the geometric distribution. Figure 9 shows the effect of monitoring multiple stocks. The y-axis now represents raw bandwidth savings, quantified by the bytes that are not transfered with respect to the strong consistency case.

5.4. Pull-Time Variations

The pull-time has a noticeable effect on performance. To measure the effect of pull time on the results, we also run

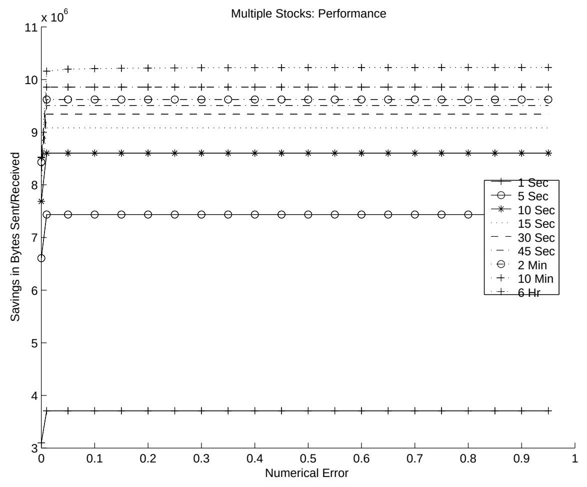

Figure 9. Bandwidth Variations-30 Stocks.

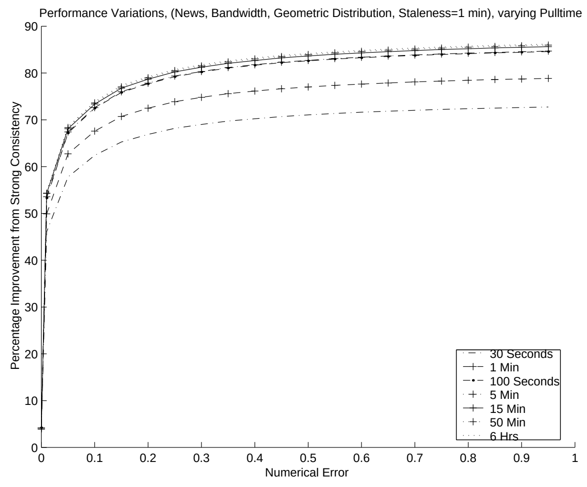

Figure 10. Sensitivity to Pull Time.

tests with pull time set to seven different values. These tests are on the Headlines service. The effect on performance is large at first for small increases in the pull time. As the pull time increases, the improvement becomes smaller. Beyond a pull time of a 100 seconds, there is little effect of pull time on performance. Figure 10 shows the performances of the seven values of pull time.

5.5. Variation in Distribution Parameters

We run simulations to test the effect of varying distribution parameters on performance. Though the data we collected in our traces largely matches a Geometric distribution, with certain parameters, this will not always be the case. We show results for the parameter P of the Geometric distribution varying from 0.1 to 0.99. We expect performance of the system to improve as the probability of zero

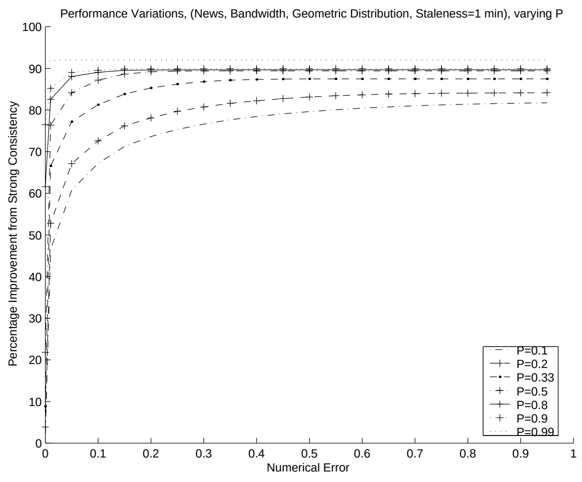

Figure 11. Sensitivity to the various Geometric distributions (varying P).

change increases. This is confirmed by a plot of various values of P for given values of staleness, shown in Figure 11. The increase in performance varies more with a change in the graph parameters for lower values of P. As we go higher, we reach the limit of improvement at around 92%.

5.6. Cost of the System

In general, our technique trades increased server memory and CPU costs for reduced bandwidth (at both server and client) and reduced energy consumption at the client. In this section, we present a brief discussion on the cost of our model (in terms of memory) at the service side. Intuitively, it is clear that the memory requirements depend on the number of clients being serviced, the number of services (headlines, stock and weather for instance) that they wish to monitor and the number of items per service that they subscribe to.

We will now analyze the memory requirements of the system. Let the system provide K services. Let each service provide I_k items. Let each item consume M_{I_k} bytes of memory. Let the number of simultaneous users be N. If each user subscribes to a set of services S_n, and keeps track of D_{S_n} items per service. Then, the memory required will be

$$M = \sum_{i=1}^{N} \sum_{j=1}^{S_i} M_j D_j$$

Now, we substitute representative figures in the equation. A reasonably popular portal may gather a million clients, so $N = 10^6$. The services deployed may be $K = 10$. On average, a user will subscribe to about five services, and track about 10 data items per service. Each data item will not cost more than 50 bytes of memory (we would need two pointers and four integers to track staleness and numerical error). Hence, we obtain a total cost of a little over 2 Gigabytes. Of course, significant assumptions go into this derivations, but this amount of memory appears reasonable, especially if the data is spread across a cluster.

6. Conclusions and Future Work

Wireless devices allow users real-time access to personalized information such as news headlines, email, stock quotes, and online auctions. Pushing all updates to such rapidly changing information is wasteful of both network bandwidth and battery power. Existing consistency models for such services allow users only to specify a coarse-grained timeout on how often information should be pushed. In this paper, we argue for the benefits of a continuous consistency model for user access to wireless portal services. Using this model, users are able to specify the maximum error in their view of the data. For instance, users may specify that they wish to receive an updated stock quote only if the value changes by 3%. Services may also use application-specific semantics to control data consistency—e.g., new bids carry more weight as the auction draws to a close. We use a simulator to model the bandwidth and energy benefits as a function of consistency and distribution of updates to the underlying data. We further use a trace based study of updates to weather, news, and stock quotes to determine potential update distributions for input to our simulator. While our trace is not necessarily representative, simulation results using these update distributions indicate significant improvements over strong consistency and over more ad hoc coarse-grained timeouts.

In the future, we intend to extend our simulator to other contexts, including sharing rapidly changing data in cooperative proxy caches and content distribution networks. Further, we are building a sample portal service that allows users to specify their consistency requirements. This will allow us to measure the computation and storage overhead associated with providing flexible consistency guarantees in the server. Further, we will be able to verify our simulation results by accessing the sample service using real wireless clients. This infrastructure will also allow us to measure any additional savings associated with reducing the number of connections to/from the server and eliminating network packets associated with TCP's connection establishment and tear-down. Finally, we wish to explore the use of more benefits of user-specified weights on incoming messages at the server. For example, users may specify that email messages coming from a particular source should be assigned a higher update weight than others. This in turn will force such important messages to be pushed to the user more quickly.

References

[1] P. Bernstein and N. Goodman. The failure and recovery problem for replicated distributed databases. In *ACM Transactions on Database Systems*, December 1984.

[2] B. Coan, B. Oki, and E. Kolodner. Limitations on database availability when networks partition. In *PODC 1986*, August 1986.

[3] S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. In *Computing Surveys*, 1985.

[4] R. Guy, J. Heidemann, T. P. J. W. Mak, G. Popek, and D. Rothmeier. Implementation of the ficus replicated file system. In *USENIX Conference Proceedings*, June 1990.

[5] R. G. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. J. Popek. Rumor: Mobile data access through op-timistic peer-to-peer replication. In *Advances in Database Technologies:ER '98*, November 1998.

[6] P. J. Keleher. Decentralized replicated object protocols. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, April 1999.

[7] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. In *Thirteenth ACM Symposium on Operating Systems Principles*, February 1992.

[8] R. Ladin, B. Liskov, L. Shirira, and S. Ghemawat. Providing high availability using lazy replication. In *ACM Transactions on Computer Systems, TOCS*, 1992.

[9] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for http. In *SIGCOMM 1997*, September 1997.

[10] B. Noble, B. Fleis, M. Kim, and J. Zajkowski. Fluid replication. In *Netstore '99, the Network Storage Symposium, Seattle, WA*, October 1999.

[11] K. Petersen, M. J.Spreitzer, D. B.Terry, M. M.Theimer, and A. J.Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16), Saint Malo, France*, October 1997.

[12] D. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly con-nected replicated storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

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

[14] H. Yu and A. Vahdat. Efficient Numerical Error Bounding for Replicated Network Services. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*, September 2000.

[15] H. Yu and A. Vahdat. Combining Generality and Practicality in a Conit-Based Continuous Consistency Model for Wide-Area Replication. In *The 21st IEEE International Conference on Distributed Computing Systems (ICDCS)*, April 2001.