AN OPTIMIZATION TECHNIQUE FOR ONLINE DATA DELIVERY

S.V.V.D Venu Gopal[1], G.M. Satyanarayana[2], B. Yugandhar[3]

[1] *Assistant professor, Dept of Computer Science and Engineering, ASR College of Engineering, A.P*
[2]*Final MTech Student,Dept of Computer Science and Engineering, ASR College of Engineering, A.P*
[3]*Assistant professor, Dept of CSE, Swarnandhra Institute of Engg & Technology, Seetharampuram, Narsapur, AP.*

Abstract

In this paper presents a framework and optimization technique , data delivery to multiple users and accessing data from anonymous servers. First approach strict setting of meeting for minimizing and maximizing the user utility. Second approach is Using real (RSS feeds) and synthetic traces, we empirically analyze the behaviour of SUP under varying conditions. A shortcoming of static solutions to pull-based delivery is that they cannot adapt to the dynamic behaviour of wide area applications. We further show that SUP can exploit feedback to improve user utility with only a moderate increase in resource utilization.

Index Terms—*Distributed databases, online information services, client/server multitier systems, online data delivery.*

----------***----------

1. INTRODUCTION

A distributed database is a database in which storage devices are not all attached to a common processing unit such as the CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely coupled sites that share no physical components.

Collections of data (e.g. in a database) can be distributed across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. The replication and distribution of databases improves database performance at end-user worksites. To ensure that the distributive databases are up to date and current, there are two processes: replication and duplication. Replication involves using specialized software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. The replication process can be very complex and time consuming depending on the size and number of the distributive databases. This process can also require a lot of time and computer resources. Duplication on the other hand is not as complicated. It basically identifies one database as a master and then duplicates that database. The duplication process is normally done at a set time after hours. This is to ensure that each distributed location has the same data. In the duplication process, changes to the master database only are allowed. This is to ensure that local data will not be overwritten. Both of the processes can keep the data current in all distributive locations.

Besides distributed database replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies' implementation can and does depend on the needs of the business and the sensitivity/confidentiality of the data to be stored in the database, and hence the price the business is willing to spend on ensuring data security, consistency and integrity.

Care with a distributed database must be taken to ensure the following:

- The distribution is transparent — users must be able to interact with the system as if it were one logical system. This applies to the system's performance, and methods of access among other things.
- Transactions are transparent — each transaction must maintain database integrity across multiple databases. Transactions must also be divided into sub-transactions, each sub-transaction affecting one database system.

There are mainly two approaches to store a relation r in a distributed database system:

A) Replication
B) Fragmentation

A) Replication: In replication, the system maintains several identical replicas of the same relation r in different sites.

- Data is more available in this scheme.
- Parallelism is increased when read request is served.

- Increases overhead on update operations as each site containing the replica needed to be updated in order to maintain consistency.(B) Fragmentation: The relation r is fragmented into several relations r_1, r_2, r_3....r_n in such a way that the actual relation could be reconstructed from the fragments and then the fragments are scattered to different locations. There are basically two schemes of fragmentation:
- Horizontal fragmentation - splits the relation by assigning each tuple of r to one or more fragments.
- Vertical fragmentation - splits the relation by decomposing the schema R of relation r.

Advantages

- Management of distributed data with different levels of transparency like network transparency, fragmentation transparency, replication transparency, etc.
- Increase reliability and availability.
- Easier expansion.
- Reflects organizational structure — database fragments are located in the departments they relate to.
- Local autonomy or site autonomy — a department can control the data about them (as they are the ones familiar with it.)
- Protection of valuable data — if there were ever a catastrophic event such as a fire, all of the data would not be in one place, but distributed in multiple locations.
- Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)
- Economics — it costs less to create a network of smaller computers with the power of a single large computer.
- Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems).
- Reliable transactions - Due to replication of database.
- Hardware, Operating System, Network, Fragmentation, DBMS, Replication and Location Independence.
- Continuous operation.
- Distributed Query processing.
- Distributed Transaction management.

Single site failure does not affect performance of system. All transactions follow A.C.I.D. property: a-atomicity, the transaction takes place as whole or not at all; c-consistency, maps one consistent DB state to another; i-isolation, each transaction sees a consistent DB; d-durability, the results of a transaction must survive system failures. The Merge Replication Method used to consolidate the data between databases.

In this paper, we address the task of targeted data delivery. Users may have specific requirements for data delivery, e.g., how frequently or under what conditions they wish to be alerted about update events or update values, or their tolerance to delays or stale information. The challenge is to deliver relevant data to a client at the desired time, while conserving system resources. We consider a number of scenarios including RSS news feeds, stock prices and auctions on the commercial Internet, and scientific data sets and Grid computational resources. We consider architecture of a proxy server that is managing a set of user profiles that are specified with respect to a set of remote autonomous servers.

2. RELATED WORK

2.1 Push-Based Content Delivery

In This model the network as a connected graph G(V,E), where V represents the set of clients (C) and servers (S), and E represents the network connections between clients and servers. G'(V,E'), the reach ability graph of G is a clique, where E' represents the edges in the reachability graph G' and each edge corresponds to the set of shortest paths between nodes in V. Note that each server may not have a full, up-to-date copy of this reachability graph. An object represents any kind of interactive data resource. It could be an application, a file, or an avatar in a virtual world. Each object has a source/home, source(ok) where ok ∈ O. The source of an object is the server node that has the original copy of the object. ingle object. A client has a number of objects that it may want to view. This set of objects is called the clients' hotset, hotset(ci), where ci ∈ C. Each client associates a priority with each object. The priority represents how important the object is to the client (Figure 3).

Updates to objects are initiated by clients. A client may update the state of any of the objects in its hotset and has to see any changes to its state made by other clients. These updates are transmitted in a push-based manner to all the other clients viewing/interacting with this object.

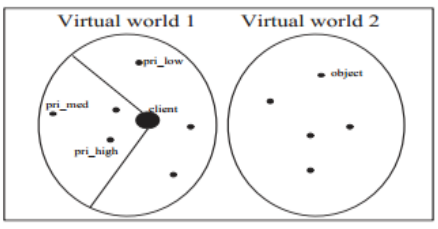


Fig. 3. Two virtual worlds, and the objects within, served by the same server. A client is viewing one of these virtual worlds; i.e., the objects in this world are in the hot set of the client. The objects are assigned priorities based on how important they are for proper visualization of the world: closer objects are given higher priorities.

Each server hosts at least one object and often more. Each server has a capacity associated with it. The value of capacity collectively represents constraints imposed on the server, in terms of memory, local bandwidth, and processing capabilities. If the load on the server exceeds the server's capacity, then it has to offload some of the services to an other server. The following steps ensure consistency among the replicated servers:
1) A client ci updates (changes the state) an object ok present on one of the replicated servers sj (the server the client is connected to).
2) The update is transmitted from server sj to the server which is the source of the object ok.
3) The update is then transmitted from the source of the object to all the servers that have a copy of the object ok.
4) Each server then transmits the update to all client that are currently connected to it and are interested in viewing updates made to the object ok. Content Replication and Client Redirection When a server is overwhelmed, a new server node needs to be selected to replicate the objects present on the overwhelmed server. Clients currently connected to this server need to be redirected to the new server. The entire process of server replication consists of the following steps
1) An overwhelmed server makes the decision to replicate.
2) A replica of the server is created at a suitable network node.
3) A portion of the clients connected to the replicated server are redirected to the newly created server.

To address some of the limitations of OptMon1, we propose a framework where we consider the dual of the previous optimization problem as follows: Given some set of user profiles, minimize the consumption of system resources while satisfying all user profiles. We label this problem as OptMon2; it will be formally defined in Section 2. With this class of problems, user needs are set as the constraining factor of the problem (and thus, need to be satisfied), while resource consumption is dynamic and changes with needs. We present an optimal algorithm in the OptMon2 class, namely, Satisfy User Profiles (SUPs). SUP is simple yet powerful in its ability to generate optimal scheduling of pull requests. SUP is an online algorithm; at each time point, it can get additional requests for resource monitoring. Through formal analysis, we identify sufficient conditions for SUP to be optimal given a set of updates to resources. Therefore, it satisfies all client needs with minimal resource consumption. We also show the conditions under which SUP can optimally solve OptMon1 as well. SUP depends on an accurate model of when updates occur to perform correctly. However, in practice, such estimations suffer from two problems. First, the underlying update model that is used is stochastic in nature, and therefore, updates deviate from the expected update times. Second, it is possible that the underlying update model is (temporarily or permanently) incorrect, and the real data stream behaves differently than expected. To accommodate changes to source behavior, compensating for stochastic behavior, correlations, and bursts, SUP exploits feedback from probes and can adapt the probing schedule in a dynamic manner and improve scheduling decisions. We also present SUP(λ) that addresses the second problem and can locally apply modifications to the update model parameters. Both SUP and SUP(_) are shown empirically to work well under stochastic variations.

We present an extensive evaluation of the solutions to the two monitoring problems. For our experimental comparison of OptMon1, we consider the WIC algorithm [24] which provides the best solution in the literature. For OptMon2, we consider the ubiquitous TTL algorithm [15]. We use real traces from an RSS server and synthetic data, and several example profiles. Our experiments show that we can achieve a high degree of satisfaction of user utility when the estimations of SUP closely estimate the real event stream, and can save significant amount of system resources compared to solutions that have to meet strict a priori allocations of system resources. We further show that feedback improves user utility of SUP(λ) with only a moderate increase in resource utilization.

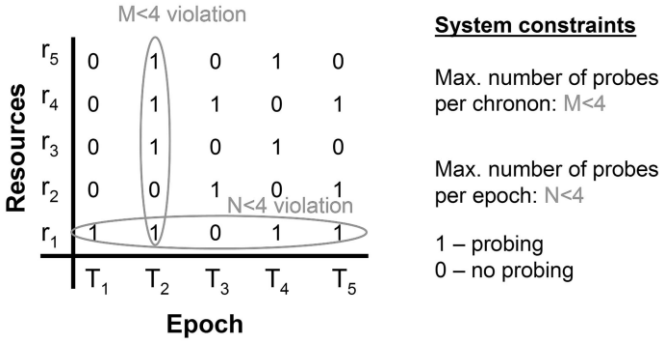


Fig. 1. Examples of a schedule and system constraints.

We now present our framework for targeted data delivery. We define a specification language for user profiles; it can be used in conjunction with push or pull based delivery. We then focus on pull-based methods and introduce dual offline optimization problems, $OptMon_1$ and $OptMon_2$ then discuss schedules and the utility of probing. Let $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ be a set of pages, taken from various page classes (e.g., eBay, Yahoo! RSS pages, etc.) We denote by $\mathcal{D}_i$ the class of page P_i .Let $\mathcal{T}$ be

an epoch and let $\{T_1, T_2, ..., T_N\}$ be a set of chronons (time points) in $\mathcal{T}$ A schedule $S = \{s_{i,j}\}_{i=1...n, j=1,...N}$ is a set of binary decision variables, set to 1 if page P_i is probed at time T_j and 0 otherwise. Let $\mathcal{S}$ be the set of all possible schedules.

2.2 User Profiles and the Monitoring Task

User profiles are declarative specifications of goals for data delivery. Our profile language consists of three parts, namely Domain, Notification, and Profile. Domain is a set of classes in which users have some interest. The domain includes two class types, namely "Query classes," classes that users wish to monitor and "Trigger classes" that are used to determine when some monitoring action should be executed. For example, the condition for monitoring a page from class D_1 (say stock prices) may be based on updates to a page from class D_2 (say, financial news reports). Query classes and trigger classes may overlap. We present here a simple domain creation example in our profile language, named *RSS Feeds*" with three fields, title, description, and publication date. A full-scale profile language discussion is beyond the scope of this paper.

CREATE DOMAIN *"RSS Feeds"* AS
CLASS::RSS(channel.item.title:String
channel.item.description:String, _
channel.item.pubDate:String);

A profile is a triplet. It contains a list of predefined domains such as a traffic domain or a weather domain, a set of notification rules that are defined over the domains (see below), and a set of users that are associated with the profile. A notification rule may be associated with different profiles. The following profile is defined for user *u025487* over the domain of *RSS feeds*, using SQL as its query language:

CREATE PROFILE *"RSS Monitoring"* AS
(DOMAIN *"RSS Feeds"*,
LANGUAGE *"SQL"*,
USER *"u025487"*);

A notification rule η is a quadruple of the form $\langle Q, Tr, \mathcal{T}, U \rangle$. Q. is a query written in any standard query language, specifying the Query classes (in some domain) that are to be monitored. *Tr* is a triggering expression and can include a triggering event and a condition that must be satisfied for monitoring to be initiated; both are defined for the Trigger class(es). $\mathcal{T}$ is the period of time during which the notification rule needs to be supported. U is a utility function specifying the utility gained from notifications of Q For each event that occurs, the notification has two possible states, namely "Triggered" and "Executable." A notification enters the Triggered state when an event specified in *Tr* occurs; the condition of *Tr* is then evaluated. If this condition is true, then the notification goes to the Executable state (for that event). A notification remains executable as long as the condition is true. The period in which a notification rule is executable was referred to in the literature as *life* [9]. Two examples of life we shall use in this paper are overwrite, in which an upade is available for monitoring only until the next update to the same page occurs. A more relaxed life setting is called *window(Y)*, for which an update can be monitored up to 2 chronons after it has occurred.

The period of time on which the notification is executable for some event defines a possible "execution interval," during which monitoring should take place. That means that the query part of a notification that defines the monitoring task should be executed. Each notification rule, η is associated with a set of execution intervals $EI(\eta)$. For each $I \in EI(\eta)$we define $\tau(I)$ as the times T_j on which the notification is executable for interval I. It is worth nothing that execution intervals of a notification rule may overlap, thus the execution of notification query may occur at the same time for two or more events that cause the notification to become executable. As an example, suppose the user would like to be notified every time there are X new RSS feeds. The notification rule Num Update Watch, to be associated with *RSS Monitoring* profile is defined as follows:

INSERT NOTIFICATION *"Num Update Watch"*
INTO *"RSS Monitoring"*
SET QUERY *"SELECT channel.item.title,channel.item.description FROM RSS"*
SET TRIGGER *"ON INSERT TO RSS*
WHEN *COUNT(*) % X =0"*
START *NOW*
END *"30 days" + NOW*
SET UTILITY *STRICT*

Monitoring can be done using one of three methods, namely push-based, pull-based, or hybrid. With push-based monitoring the server pushes updates to clients, providing guarantees with respect to data freshness at a possibly considerable overhead at the server. With pull-based monitoring, content is delivered upon request, reducing overhead at servers, with limited effectiveness in estimating object freshness. The hybrid approach combined push and pull, either based on resource constraints [6] or role definition. For the latter, consider the user profile language we have presented. Here, it is possible that servers of trigger classes will push data to clients, while data regarding query classes will be monitored by pulling

content from servers once a notification rule is satisfied. As another example for the hybrid approach, consider a three-layer architecture, in which a mediator is positioned between clients and servers. The mediator can monitor servers by periodically pulling their content, and determine when to push data to clients based on their content delivery profiles. In the rest of the paper we focus on challenges in pullbased monitoring. We start with detailing the impact on pull-based monitoring on clients and introduce the dual optimization problem.

Framework of Dual Offline Optimization Problems

There are two, principally different, approaches to support pull-based targeted data delivery. One formulation $OptMon_1$ assumes an a priori independent assignment of system resources to this task, e.g., an upperbound on bandwidth for probing. The task is to maximize user benefit under such constraints. It is as follows:
maximize user utility

_ _
s.t satisfying system constraints

For example, in [9] $OptMon_1$involves a system resource constraint of M the maximum number of probes per chronon for all pages in P . One can specify system resources in terms of number of probes, assuming each probe has an overhead of opening a TCP/IP channel of communication, downloading information from the server, deciding on the timing of the next probe, etc. User utility can be parameterized to represent the amount of tolerance a user has to delayed delivery. We propose a dual formulation $OptMon_2$, , which reverses the roles of user utility and system constraints. It assumes that the system resources that will be consumed to satisfy *user profiles* should be determined by the specific profiles and the environment, e.g., the model of updates. Thus, we do not assume an a priori limitation of system resources. $OptMon_2$ is the following optimization problem:

minimize system resource usage

_ _
s.t satisfying user profiles

The dual problems are inherently different. To illustrate this, consider two resource constraints, namely M (the maximum number of probes per chronon) and N , the number of chronons in $\mathcal{T}$. The total number of available probes in $\mathcal{T}$ is N. M . The behavior of all solutions to $OptMon_1$ will be controlled by the values for M and N . For example, a choice of a smaller chronon size results in larger N and an increased utilization of probes (system resources) b y any solution to $OptMon_1$. On the other hand, solutions To $OptMon_2$ can benefit from parameter settings, but the parameter values do not control their behavior. For example, a smaller chronon size and larger N allows a solution to $OptMon_2$ to probe resources at a finer level of chronon granularity; however, the minimization of resource utilization would ensure that resource consumption will not increase in vain. To summarize, solutions to the two problems cannot be compared directly. No solution for one problem can dominate a solution to the other, for all possible problem instances.

2.3 Roadmap for Investigating the Dual Problem

In Section 3 we introduce SUP, an efficient algorithm for the challenge of targeted data delivery, given a user profile. SUP is guaranteed to minimize system resources while satisfying a user profile. SUP is an offline algorithm. It determines a probing schedule given an a priori update model of resources. Therefore, in run-time it may not be optimal due to two main problems. First, an update model for pull-based monitoring is necessarily stochastic and therefore is subject to variations, due to the model variance. Second, it is possible that the update model is inaccurate to start with, which means that replacing it with another, more accurate update model would yield better schedule. In this research, we address the first problem by offering *fbSUP*, an online algorithm that makes use of feedback to tune its schedule. We defer the algorithmic solution of the second problem to an extended version of this work.

2.4 Schedules and the Utility of Probing

Let $\mathcal{N}_k$ be the set of notification rules of profile p_k. Let $\eta \in \mathcal{N}_k$ be a notification rule that utilizes classes from p_k domain and let Q^η be the set of all pages in $\mathcal{P}$ that are in the domain of p_k. We now define the satisfiability of a schedule with respect to η as follows:

Definition 2.1. *Let $S \in \mathcal{S}$ be a schedule, η be a notification to* satisfy *η in $\mathcal{T}$ (denoted $S \models_{\mathcal{T}} \eta$) if $\forall I \in EI(\eta) \forall P_i \in$ rule with Q^η, and $\mathcal{T}$ be an epoch with N chronons. S is said* $Q^\eta(\exists T_j \in \tau(I) : s_{i,j} = 1)$.

Definition 2.1 requires that in each execution interval, every page in Q^η _ is probed at least once. Whenever it becomes clear from the context, we use $S \models \eta$ instead of $S \models_{\mathcal{T}} \eta$ This definition is easily extended to a profile and a set of profiles, as follows:

Definition 2.2. *Let $S \in \mathcal{S}$ be a schedule, $\{p_1, p_2, ..., p_m\}$ be a set of profiles, and $\mathcal{T}$ be an epoch with N chronons. p_k) if for each notification rule $\eta \in \mathcal{N}_k$, $S \models \eta$. S is said to* satisfy *$p_k \in \{p_1, p_2, ..., p_m\}$ (denoted $S \models$ S is said to* satisfy *$\{p_1, p_2, ..., p_m\}$ (denoted $S \models$ $\{p_1, p_2, ..., p_m\}$) if for each profile $p_k \in \{p_1, p_2, ..., p_m\}$ $S \models p_k$.*

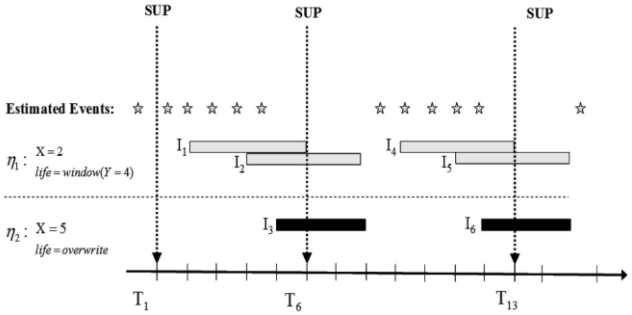
Given a notification rule $\eta \in \mathcal{N}_k$ and a page a $P_i \in \cup_{\eta \in \mathcal{N}_k} Q^\eta$, utility function $u(P_i, \eta, T_j)$ describes the utility of probing a page P_i at chronon T_j. Intuitively, probing a page P at time T is useful (and therefore should receive a positive utility value) if it belongs to a class that is referred to in the Query part of the notification rule and if the condition in the Trigger part of that profile holds. It is important to emphasize again the difference of roles between the Query part and the Trigger part of the profile. In particular, probing a page P is useful only if the data required by P. $u(P_i, \eta, T_j)$ is a profile (specified in the Query part) can be found at is derived by assigning positive utility when a condition is satisfied, and a utility f(otherwise. _is defined to be *strict* if it satisfies the following condition:

$$u(P_i, \eta, T_j) = \begin{cases} w & T_j \in \cup_{I \in EI(\eta)} \tau(I) \wedge P_i \in Q^\eta \\ 0 & \text{otherwise} \end{cases}$$

From now on we shall assume binary utility, i.e. Example of strict utility functions include [9], *uniform* (where utility is independent of delay) and *sliding window* (where utility is 1 within the window and 0 out of it). Examples of non strict utility functions are non-linear decay functions. For simplicity, we shall restrict ourselves to strict utility functions.

3 .THE SUP ALGORITHM

Recall that a notification rule η is associated with a set of pages Q^η Given a notification rule and the set of its execution intervals $EI(\eta)$, SUP identifies the set of pages $Q_I^\eta \subseteq Q^\eta$ that must be probed in an execution interval I.We present a static algorithm SUP for solving $OptMon_1$. By static we mean that the schedule is determined a-priori.



Illustrating example of SUP execution.

Algorithm 1 (SUP)
Input: $\mathcal{P}, \mathcal{T}, \mathcal{N} = \cup_{l=1}^{m} \mathcal{N}_l$
Output: $S = \{s_{i,j}\}$
(1) For all pages $P_i \in \mathcal{P}$ and chronons $T_j \in \mathcal{T}$:
(2) Initialize $s_{i,j} \leftarrow 0$.
(3) For $l = 1$ to $|\mathcal{N}|$:
(4) $T_l = \min_{I \in EI(\eta)} \{\max \tau(I)\}$
/* T_l, is the last chronon of the first */
/* execution interval of notification rule η_l */
(5) repeat
(6) $\tau_j = \min_{l=1}^{|\mathcal{N}|}(T^l)$
/* τ_j is the earliest chronon for which a notification rule */
/* is executable and when SUP will probe*/
(7) $\eta = argmin_{l=1}^{|\mathcal{N}|}(T^k)$
/* η is the notification rule whose */
/* pages in Q_I^η need to be probed*/
(8) $I = argmin_{I \in EI(\eta)} \{\max \tau(I)\}$
(9) For all $P_i \in Q_I^\eta$:
(10) set $s_{i,j} \leftarrow 1$
(11) For $k = 1$ to $|\mathcal{N}|$:
(12) *UpdateNotificationEIs*$(\tau_j, \mathcal{N}_k)$
(13) $T_l = \min_{I \in EI(\eta)} \{\max \tau(I)\}$
(14) until $T^l > N$

The algorithm builds a schedule $(\forall s_{i,j} \in S, s_{i,j} = 0)$ iteratively. It starts with an empty schedule (and repeatedly adds probes. The "for loop" in lines 3-4 generates an initial probing schedule, where the last chronon in the first $I \in EI(\eta)$ is picked to execute the probe. Lines 5-8 determine the earliest chronon in which a probe is to be made, the notification rule associated with this probe, and the specific execution interval. All pages that belong to classes in the query part of that notification rule are probed (lines 9-10). In line 12, the algorithm uses a routine, *UpdateNotificationEIs*, Let $l = \eta$ to ensure that pages that belong to overlapping intervals are only probed once. Let <_ , be the assignment in line 7 of the algorithm. $\eta$is the notification rule whose execution interval I is processed at time j, and all pages that belong to classes in Q_I^η. __at time are scheduled for probing. Given an execution interval I' of a notification rule η' , this routine removes from $Q_{I'}^{\eta'}$ the (possibly empty) class set $Q_I^\eta \cap Q_{I'}^{\eta'}$ if $\tau(I) \cap \tau(I') \neq \emptyset$. By doing so, we ensure that pages that belong to overlapping execution intervals I will be probed only once. In addition, this routine removes any execution interval for which $Q_I^\eta = \emptyset$, , allowing lines 4 and 13 to consider only execution intervals for which monitoring is still needed. The process continues until the end of the epoch. Generally speaking, a new probe is set for a page at the last possible chronon where a notification remains executable. That is, it is deferred to the last possible chronon where the utility is still 1. This, combined with the use of the routine *UpdateNotificationEIs* is needed to develop an optimal schedule, in terms of resource utilization.

CONCLUSION

For many applications to support diverse files across multiple sources and reduce resource consumption. For pull based there is a need to minimize number probes and improve scalability. Solutions that can adapt to changes in source behavior are also important due to the difficulty of predicting when updates occur. In this paper, we have addressed these challenges through the use of a new formalism of a dual optimization problem (OptMon2), reversing the roles of user utility and system resources. This revised specification leads naturally to a surprisingly simple, yet powerful algorithm (SUP) which satisfies user specifications while minimizing system resource consumption.

We have formally shown that SUP is optimal for $OptMon_2$ and under certain restrictions can be optimal for $OptMon_1$ as well. We have empirically shown, using data traces from diverse Web sources that SUP can satisfy user profiles and capture more updates compared to existing policies. We have also analyzed the impact of profiles, life parameters, and update models on SUP online performance.

REFERENCES

[1] A. Adi and O. Etzion, "Amit—The Situation Manager," Int'l J. Very Large Data Bases, vol. 13, no. 2, pp. 177-203, May 2004.
[2] L. Bright, A. Gal, and L. Raschid, "Adaptive Pull-Based Policies for Wide Area Data Delivery," ACM Trans. Database Systems, vol. 31, no. 2, pp. 631-671, 2006.
[3] L. Bright and L. Raschid, "Using Latency-Recency Profiles for Data Delivery on the Web," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 550-561, Aug. 2002.
[4] D. Carney, S. Lee, and S. Zdonik, "Scalable Application-Aware Data Freshening," Proc. IEEE CS Int'l Conf. Data Eng., pp. 481-492, Mar. 2003.
[5] L.S. Chandran, L. Ibarra, F. Ruskey, and J. Sawada, "Generating and Characterizing the Perfect Elimination Orderings of a Chordal Graph," Theoretical Computer Science, vol. 307, no. 2, pp. 303-317, 2003.
[6] M. Cherniack, E. Galvez, M. Franklin, and S. Zdonik, "Profile- Driven Cache Management," Proc. IEEE CS Int'l Conf. Data Eng., pp. 645-656, Mar. 2003.
[7] J. Cho and H. Garcia-Molina, "Synchronizing a Database to Improve Freshness," Proc. ACM SIGMOD, pp. 117-128, May 2000.
[8] J. Cho and A. Ntoulas, "Effective Change Detection Using Sampling," Proc. Int'l Conf. Very Large Data Bases (VLDB), 2002.
[9] "CNN Top Stories RSS Feed," http://rss.cnn.com/services/rss/ cnn_topstories.rss, 2010.
[10] E. Cohen and H. Kaplan, "Refreshment Policies for Web Content Caches," Proc. IEEE INFOCOM, pp. 1398-1406, Apr. 2001.
[11] U. Dayal et al., "The HiPAC Project: Combining Active Databases and Timing Constraints," SIGMOD Record, vol. 17, no. 1, pp. 51-70, Mar. 1988.
[12] P. Deolasee, A. Katkar, P. Panchbudhe, K. Ramamritham, and P. Shenoy, "Adaptive Push-Pull: Disseminating Dynamic Web Data," Proc. Int'l World Wide Web Conf. (WWW), pp. 265-274, May 2001.
[13] J. Eckstein, A. Gal, and S. Reiner, "Optimal Information Monitoring under a Politeness Constraint," Technical Report RRR 16-2005, RUTCOR, Rutgers Univ., May 2005.
[14] A. Gal and J. Eckstein, "Managing Periodically Updated Data in Relational Databases: A Stochastic Modeling Approach," J. ACM, vol. 48, no. 6, pp. 1141-1183, 2001.
[15] J. Gwertzman and M. Seltzer, "World Wide Web Cache Consistency," Proc. USENIX Ann. Technical Conf., pp. 141-152, Jan. 1996.
[16] "BlackBerry Wireless Handhelds," http://www.blackberry.com, 2010.
[17] Z. Jiang and L. Kleinrock, "Prefetching Links on the WWW," Proc. IEEE Int'l Conf. Comm., 1997.
[18] G. Kappel, S. Rausch-Schott, and Retschitzegger, "Beyond Coupling Modes: Implementing Active Concepts on Top of a Commercial OODBMS," Object-Oriented Methodologies and Systems, S. Urban and E. Bertino, eds., pp. 189-204. Springer-Verlag, 1994.
[19] J.-J. Lee, K.-Y. Whang, B.S. Lee, and J.-W. Chang, "An Update-Risk Based Approach to TTL Estimation in Web Caching," Proc. Conf. Web Information Systems Eng. (WISE), pp. 21-29, Dec. 2002.
[20] C. Liu and P. Cao, "Maintaining Strong Cache Consistency on the World Wide Web," Proc. Int'l Conf. Distributed Computing Systems (ICDCS), 1997.
[21] H. Liu, V. Ramasubramanian, and E.G. Sirer, "Client and Feed Characteristics of rss, a Publish-Subscribe System for Web Micronews," Proc. Internet Measurement Conf. (IMC), Oct. 2005.
[22] C. Olston and J. Widom, "Best-Effort Cache Synchronization with Source Cooperation," Proc. ACM SIGMOD, pp. 73-84, 2002.
[23] V. Padmanabhan and J. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," ACM SIGCOMM Computer Comm. Rev., vol. 26, no. 3, pp. 22-36, July 1996.
[24] S. Pandey, K. Dhamdhere, and C. Olston, "WIC: A General- Purpose Algorithm for Monitoring Web Information Sources," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 360-371, Sept. 2004.

[25] "Promo Language Specification," http://ie.technion.ac.il/~avigal/ProMoLang.pdf, 2010.
[26] H. Roitman, A. Gal, and L. Raschid, "Capturing Approximated Data Delivery Tradeoffs," Proc. IEEE CS Int'l Conf. Data Eng., 2008.
[27] "RSS," http://www.rss-specifications.com, 2010.
[28] J.L. Wolf, M.S. Squillante, P.S. Yu, J. Sethuraman, and L. Ozsen, "Optimal Crawling Strategies for Web Search Engines," Proc. Int'l World Wide Web Conf. (WWW), pp. 136-147, 2002.
[29] E. Yashchin, "Change-Point Models in Industrial Applications," Nonlinear Analysis, vol. 30, pp. 3997-4006, 1997.
[30] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar, "Engineering Server- Driven Consistency for Large Scale Dynamic Web Services," Proc. Int'l World Wide Web Conf. (WWW), pp. 45-57, May 2001.