Timeliness-Accuracy Balanced Collection of Dynamic Context Data

Qi Han, *Member*, *IEEE*, and Nalini Venkatasubramanian, *Member*, *IEEE*

Abstract—In the future, we are likely to see a tremendous need for context-aware applications which adapt to available context information such as physical surroundings, network, or system conditions. We aim to provide a fundamental support for these applications—a real-time context information collection service. This service delivers the right context information to the right user at the right time. The complexity of providing the real-time context information service arises from 1) the dynamically changing status of information sources, 2) the diverse user requirements in terms of data accuracy and service latency, and 3) constantly changing system conditions. In this paper, we take into consideration these dynamics and focus on addressing the trade-offs between timeliness, accuracy, and cost for information collection in distributed real-time environments. We propose a middleware-based approach to enable a judicious composition of services for accuracy-aware scheduling and cost-aware database maintenance. Specifically, we characterize the problem in terms of Quality-of-Service Satisfaction (QoSSat), Quality-of-Data Satisfaction (QoDSat), and Cost. We propose a middleware framework for the real-time information collection process, where the information mediator coordinates and facilitates communication between information sources and consumers. We design a family of algorithms for real-time request scheduling, request servicing, and directory service maintenance to be implemented at the mediator to support QoSSat and QoDSat. Our studies indicate that the composition of our proposed scheduling algorithm and directory service maintenance policy can improve the overall efficiency of the system. We also observe that the proposed policies perform very well as the system scales in the number of information sources and consumer requests.

Index Terms—Distributed systems, context-sensitive middleware, real-time, quality of service, quality of data.

1 Introduction

Recent advances in mobile computing, communications, and embedded systems are likely to enable the deployment of large-scale ubiquitous computing environments. We expect that real-time context information services will gain importance in the future. Examples of applications that require real-time access to distributed information include network management, stock trading, air traffic control, security/surveillance, medical alerts, and patient tracking. To facilitate such real-time applications, an information collection architecture that can seamlessly provide real-time access to dynamically changing data regardless of diverse user requirements and changing system conditions is a must.

Real-time information collection in a dynamic environment presents the system designer with interesting challenges. First, information sources provide a continuous stream of data that can dynamically vary over time. This information may need to be captured and stored rapidly and accurately. Second, users requiring access to this dynamically changing data present variable user requirements in terms of the accuracy of data and timeliness of the service. Furthermore, network and service providers would like to ensure effective utilization of underlying computation, communication, and storage resources. Ideally, applications would like to obtain accurate state information in a real-time manner with the least cost. The underlying middleware architecture must deal with the issue of balancing these competing goals of timeliness, accuracy, and cost-effectiveness. Since many applications are willing to tolerate information imprecision and bounded delivery latencies, our strategy is to exploit these accuracy and latency margins to ensure that most applications receive information at the desired levels of quality and timeliness while minimizing resource consumption.

Given the data intensive nature of the system, an information repository, i.e., directory service (DS), is a must for the system to function efficiently. Maintaining an effective DS, which reflects the data changes as closely as possible/necessary, is not a trivial task in dynamic environments. In addition to processing source updates, the system must also respond to user requests in a timely fashion. Processing source updates at the expense of user requests will mean that fewer user requests will finish on time, while delaying source updates in favor of user requests will mean that the DS will not be representative of the state of the external environment. In addition to timeliness constraints, user requests may also provide accuracy constraints. Since information in the DS may not be up to date, there is a possible mismatch between the DS accuracy and user accuracy constraints. We are faced with a cost-accuracy trade-off since an accurate DS makes it easier to satisfy more user requests with less overhead in a more timely fashion, but could also introduce a high DS maintenance cost.

Contributions of this Paper. Existing and ongoing research in data management addresses the trade-offs between timeliness and data freshness, as well as the

- *Q. Han is with the Department of Mathematical and Computer Sciences, Colorado School of Mines, Golden, CO 80227. E-mail: qhan@mines.edu.*
- *N. Venkatasubramanian is with the Department of Computer Science, University of California, Irvine, Irvine, CA 92697. E-mail: nalini@ics.uci.edu.*

Manuscript received 22 Apr. 2004; revised 1 Dec. 2005; accepted 31 Mar. 2006; published online 27 Dec. 2006.
Recommended for acceptance by X. Zhang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0104-0404.

TABLE 1
Terms and Symbols Used in the Paper

Term	*Meaning*	*Term*	*Meaning*
POP_s	popularity of source s	$Bias_{cr}$	preference of consumer request cr
PER_r	periodicity of request r	TT_{cr}	total time taken to complete consumer request cr
UR_r	urgency of request r	$FI(A_{cr})$	fidelity of answer A for consumer request cr
RDL_r	relative deadline of request r	$FI_{db}(s,T)$	DS fidelity of source s over time duration T
ADL_r	absolute deadline of request r	$AFI_{db}(S,T)$	aggregate DS fidelity over time duration T
(L,U)	(lower,upper) bound of a range	$VA_{db}(s,t)$	DS validity for source s at time t
V	real value of the source	$VA_{db}(s,T)$	DS validity for source s over time duration T
$PREC(L,U)$	precision of range [L,U]	$AVA_{db}(S,T)$	aggregate DS validity over time duration T

accuracy and cost trade-offs in stream-based environments. However, increasingly many applications need more fine-grained interactions with the system in the format of multiple individual information retrieval requests rather than transactions. In addition, both user requests and data source updates may have explicit time constraints and data accuracy requirements. In this context, the problem of balancing the trade-offs between 1) timeliness, 2) accuracy, and 3) cost simultaneously has not been addressed to the best of our knowledge. The mere application of previously proposed techniques in the area of real-time scheduling or dynamic data management is insufficient to address the issue at hand; a simplistic composition of the proposed techniques cannot address the cost/accuracy/timeliness trade-offs appropriately either. Therefore, we develop a middleware-based approach to address the challenges in enabling a judicious composition of these techniques. Specific contributions of this paper are as follows:

- We characterize the problem of cost-effectively providing timeliness and accuracy in terms of Quality-of-Service Satisfaction (QoSSat), Quality-of-Data Satisfaction (QoDSat), and Cost.
- We propose a middleware framework for the real-time information collection process and design a family of algorithms to support QoSSat and QoDSat. We design a Timeliness and Accuracy Balanced Scheduling (TABS) algorithm to support timely response to both user requests and source updates while ensuring that accuracy constraints of requests are satisfied. Furthermore, we present a Minimized Cost DS maintenance algorithm (MC) that balances the distribution of source update and consumer retrieval requests to maintain a constant frequency ratio. Finally, request servicing algorithms are designed to determine the process flow for each single request by analyzing the possibility of meeting its timeliness and accuracy constraints.
- Through extensive experiments, we study the interaction between scheduling, request servicing, and DS maintenance and explore dynamic adaptations and a judicious composition of the policies that address timeliness/accuracy/cost trade-offs under varying conditions.

The rest of this paper is organized as follows: Section 2 formulates the accuracy-driven real-time information collection problem. Section 3 presents a middleware architecture for real-time information collection. Section 4 addresses a solution via the algorithms for real-time request scheduling, request servicing, and DS maintenance. Section 5 presents experimental results. Section 6 discusses related work. We conclude in Section 7 with future research directions.

2 Problem Formulation

In this section, we describe data, request models for highly dynamic environments, and characterize Quality of Service Satisfaction (QoSSat) and Quality of Data (QoDSat). These definitions are used to crystalize the goals of the system, guide the design of the algorithms, and evaluate system performance. Using the notions of QoSSat and QoDSat, we provide a definition of the real-time information collection problem. For the time being, we assume the presence of a single logically centralized DS. We also assume that users only ask for current data, not historical data, at a specified time instant. Table 1 summarizes the meanings of the symbols used in the paper.

The system consists of a number of data sources which correspond to various components in a distributed infrastructure. In other words, they are managed entities, such as servers, links, sensors, or mobile/fixed hosts. They can be programmed to send updates periodically or send updates when something abnormal occurs. The information obtained from sources includes sensor data, network parameters (such as residual link bandwidth, end-to-end delay on links, etc.), server parameters (such as CPU utilization, buffer capacity, disk bandwidth, etc.), and mobile host parameters (such as mobile host location, connectivity, power level, etc.). We specifically consider systems where state information changes rapidly, generating a large number of updates. Each source has a current instantaneous value V, while its representation in the DS is a range $R: [L,U]$ with L as the lower bound, U as the upper bound, and $L \leq V \leq U$. The range is refreshed with updates (write only) to the DS from the source and queries (read only) from the DS by consumers. Data from certain sources are often accessed more frequently, so we define the popularity of source s (POP_s) as the ratio of the number of requests accessing s to the total number of requests.

Our system includes two types of requests: source update requests and consumer requests. We first define several auxiliary parameters (periodicity, urgency, relative deadline, and range precision). These parameters characterize timeliness and accuracy constraints of incoming requests. The *periodicity PER* of request r is defined as 0 if the request is aperiodic; otherwise, it is p ($p > 0$), which

indicates that r arrives every p time units. To provide more flexibility to applications, we provide two parameters (urgency and relative deadline) to specify the timeliness constraints. The urgency of a request is a coarse-level qualitative description of how quickly the request must be processed; the relative deadline of a request specifies a maximum duration the request can take. An application may specify one or both of these parameters (i.e., urgency and relative deadline); when both are present, the relative deadline takes precedence. The *urgency UR* of request r is defined as 0, 1, or 2, respectively, when the urgency is low, medium, or high. The *relative deadline RDL* of request r is defined as 0 if there is no deadline for r; otherwise, it is d $(d > 0)$, which indicates that r should be finished in d time units. The *range precision* is defined as the reciprocal of the range size, i.e., $PREC(L,U) = \frac{1}{U-L}$ [1].

Using the notions defined above, we next define source update request and consumer request. Source update requests are write-only requests to reflect the current status of the real-world.

Definition 1. *A **Source Update Request** sr is a tuple of six elements: source s, request issue time t, real value V, periodicity PER, urgency UR, and relative deadline RDL, i.e., $sr :=< s, t, V, PER, UR, RDL >$.*

Once a source update request is applied, the DS is updated with L and U. The current value V lies at the center of the range R, i.e., $U = V + R/2$ and $L = V - R/2$. To accommodate heterogeneity of source intelligence, we assume that sources are not aware of the ranges R stored in the DS; instead, the DS maintenance module is responsible for adjusting and storing the ranges.

Information consumers are application and system level tasks that use the data collected from the information sources. For instance, resource provisioning consumes information about network and system status to perform admission control and resource allocation; traffic monitoring applications obtain data from highway sensors periodically to assist in traffic planning and routing. Consumer requests are read-only queries to retrieve current values of data sources. Each consumer request is accompanied by a time constraint and also a precision constraint specifying the maximum acceptable width of the result. Since simultaneous satisfaction of both timeliness and accuracy constraints may not be feasible, we introduce the attribute *Bias* to indicate the preference of consumers in the event that both constraints cannot be met. $Bias_{cr}$ of 0, 1, or 2, respectively, indicates that consumer request cr has no preference, favors timeliness, or favors accuracy.

Definition 2. *A **Consumer Request** cr is defined as a tuple of the value of the desired source s, request issue time t, periodicity PER, urgency UR, relative deadline RDL, and accuracy requirements $PREC$, Bias factor, and*

$$cr :=< s, t, PER, UR, RDL, PREC, Bias > .$$

In order to evaluate system performance, we define QoS Satisfaction (QoSSat) and QoD Satisfaction (QoDSat) metrics as follows.

2.1 Defining the QoS Satisfaction (QoSSat) Metric

Traditional QoS requirements are specified by each individual user. QoSSat, on the other hand, captures the overall system performance by calculating the ratio of successful requests. Furthermore, QoSSat is defined over multiple dimensions: It integrates both timeliness and accuracy into one metric. QoSSat measures how effectively the system handles consumer requests and is defined as the probability of successful consumer requests. We characterize a successful consumer request cr by *timeliness (deadline) satisfaction* and *accuracy satisfaction*. The timeliness satisfaction is met if the deadline of a consumer request cr is met, i.e., $TT_{cr} \leq RDL_{cr}$, where TT_{cr} is the total time taken to complete cr. The accuracy satisfaction is measured by 1) the answer precision $PREC_{cr}$, i.e., $PREC_A \geq PREC_{cr}$, and 2) the answer fidelity, i.e., the current source value lies within the returned range. The fidelity of answer $A : [L, U]$ for consumer request cr is denoted by $Fidelity(A_{cr})$ and defined as 1 if the current source value falls inside the answer range (i.e., $L \leq V(cr.s, cr.t) \leq U$, where $V(cr.s, cr, t)$ is the current value of source s at t), otherwise it is 0.

Typically, a successful consumer request without bias meets both timeliness and accuracy satisfaction. If a consumer request indicates a bias toward timeliness/accuracy, we consider the consumer request to be successful if the deadline/accuracy is met. Therefore, the weighted number of successful requests N_{suc} can be computed as in (1), where TT_{cr_j} is the total processing time of consumer request cr_j, RDL_{cr_j} is the desired deadline of cr_j, $PREC_{A_j}$ is the precision of answer of cr_j, and $PREC_{cr_j}$ is the desired precision of cr_j.

$$\begin{aligned} N_{suc} = w_1 \cdot \|\{cr_j | Bias_{cr_j} = 0, TT_{cr_j} \leq RDL_{cr_j}, \\ Fidelity(A_{cr_j}) = 1, PREC_{A_{cr_j}} \geq PREC_{cr_j}\}\| \\ + w_2 \cdot \|\{cr_j | Bias_{cr_j} = 1, TT_{cr_j} \leq RDL_{cr_j}\}\| \\ + w_3 \cdot \|\{cr_j | Bias_{cr_j} = 2, Fidelity(A_{cr_j}) == 1, \\ PREC_{A_j} \geq PREC_{cr_j}\}\|. \end{aligned} \quad (1)$$

Definition 3. *QoS Satisfaction (QoSSat) can be defined as in (2):*

$$QoSSat = \frac{N_{suc}}{\sum_{i=0}^{2} w_i \cdot \|\{cr_j | Bias_{cr_j} = i|\}\|}. \quad (2)$$

Since QoSSat is maximized when both deadline and accuracy constraints are met, we set w_1 to be greater than w_2 or w_3; in addition, the ideal case is $QoSSat = 1$, i.e., all the requests satisfy their bias (if any). In our evaluation, we set $w_1 = 0.5$ and $w_2 = w_3 = 0.25$.

2.2 Defining the QoD Satisfaction (QoDSat) Metric

The concept of QoD has been proposed to capture how closely the cached data matches the changes in the real data [2], which is only one perspective of data quality. Another ignored aspect is the matching between cached data and required data. QoDSat integrates these two aspects and provides a comprehensive way to measure data quality. QoDSat measures how "good" the data is and varies from one system to another. In our system, the QoDSat is measured by data accuracy. We characterize data accuracy by *DS fidelity* and *DS validity*. The DS fidelity measures the divergence between a stored range in the DS and the current source value. In contrast, the DS validity with respect to a consumer request compares the precision of the stored range of the data with the precision expectation of

the consumer request accessing the data. A "good" or accurate DS maintains a range that not only reflects the current source status (i.e., it is faithful), but also satisfies the precision expectation of consumers (i.e., it is valid).

The DS fidelity of source s with current value V at time t and stored DS interval (L, U) is referred to as $FI_{ds}(s,t)$. It is 1 if $L \leq V \leq U$; otherwise, it is 0. Therefore, the DS fidelity of source s over a certain time period $T = [t_i, t_j]$ is defined as follows:

$$FI_{ds}(s,T) = FI(s,[t_i,t_j]) = \frac{1}{T} \times \int_{t_i}^{t_j} FI(s,t)dt.$$

This is equal to the fraction of time during T that s is faithful. If we assume that s is uniformly accessed during T, then the probability of accessing a fresh value of s $p_{fi}(s,T)$ is equal to the fraction of time that s is faithful [2]. Hence, we can define the aggregate DS fidelity over all sources during the entire time period T as below:

$$\begin{aligned} AFI_{ds}(S,T) &= \sum\nolimits_{s_i \in S} p_{access}(s_i) \times p_{fi}(s_i,T) \\ &= \sum\nolimits_{s_i \in S} p_{access}(s_i) \times FI_{ds}(s_i,T), \end{aligned}$$

where $p_{access}(s_i)$ is the access ratio of s_i (the ratio of the number of consumer requesting s_i to the total number of consumer requests) and $\sum_{s_i \in S} p_{access}(s_i) = 1$. Note that, when the system performance varies over time, focusing on a narrower time interval for T would allow applications to tune their responsiveness to such changes.

The DS validity for consumer request cr_i accessing s at time t measures whether the DS precision of source s meets the cr_i's precision expectation at t. Assuming the stored DS interval is (L, U), then the DS validity $VA_{ds}(cr_i(s,t))$ is defined to be 1 if $PREC(L,U) \geq PREC_{cr_i}$. Otherwise, it is 0. If there are $k > 0$ consumer requests accessing s at time t, then the DS validity for source s at time t is

$$VA_{ds}(s,t) = \frac{\Sigma_{i=1}^{k} VA_{ds}(cr_i(s,t))}{k}.$$

Therefore, if there are k_s consumer requests accessing s over a certain time period $T = [t_i, t_j]$, the DS validity of source s over T is defined as follows:

$$VA_{ds}(s,T) = VA(s,[t_i,t_j]) = \frac{\Sigma_{u=1}^{k_s} \Sigma_{v=t_i}^{t_j} VA_{ds}(cr_u(s,t_v))}{k_s}.$$

This is exactly the probability of accessing a valid value of s during T $p_{va}(s,T)$. Therefore, the aggregate DS validity can be defined as follows:

$$\begin{aligned} AVA_{ds}(S,T) &= \sum\nolimits_{s_i \in S} p_{access}(s_i) \times p_{va}(s_i,T) \\ &= \sum\nolimits_{s_i \in S} p_{access}(s_i) \times VA_{ds}(s_i,T), \end{aligned}$$

where $p_{access}(s_i)$ is the access ratio of s_i (the ratio of the number of consumer requesting s_i to the total number of consumer requests).

As stated before, DS accuracy is the combination of DS fidelity and DS validity.

Definition 4. *The Quality-of-Data Satisfaction (QoDSat) or Aggregate DS Accuracy $p_{aac}(S)$ is defined as follows:*

$$QoDSat = p_{aac}(S,T) = AFI_{ds}(S,T) \times AVA_{ds}(S,T).$$

In addition, we use $Cost$ to represent the communication overhead involved in the process of serving all the requests. It is measured as the average number of messages exchanged per request. The messages include all those exchanged for maintaining the DS and for additional probing that may service the accuracy requirements of consumer requests.

The Timeliness-Accuracy Balanced Data Collection Problem. Given a set of l sources $S = \{s_1, s_2, \ldots, s_l\}$ and an input instance (request set) I, which is a collection of m incoming source update requests and n consumer requests $I = SR \cup CR = \{sr_1, sr_2, \ldots, sr_m; cr_1, cr_2, \ldots, cr_n\}$, our objectives are to

- increase the overall QoSSat indicated by the probability of successful consumer requests that meet deadline and/or accuracy requirements,
- increase the overall QoDSat indicated by the probability of accessing accurate data in the DS, and
- decrease the overall Cost involved in the whole collection process.

3 A Middleware-Based Approach

In practice, due to highly dynamic system and network conditions, unpredictable application workloads, and frequently changing information sources, the joint satisfaction of QoSSat, QoDSat, and Cost is very complicated. In fact, the interrelationship between QoSSat and QoDSat is not straightforward. One may argue that, by maintaining an accurate DS (i.e., improving QoDSat), the number of deadline meets can be increased (i.e., improving QoSSat is achieved) since additional probes are avoided. A natural approach to improve QoDSat is to give a higher priority to the source update request. However, this may result in an increase in the number of missed deadlines and consequently decreasing QoSSat. A simplistic composition of existing techniques cannot address the cost/accuracy/timeliness trade-offs appropriately.

- While real-time scheduling algorithms determine the order of request processing with the objective of increasing the number of requests with their time constraints met, they are incognizant of communication and data processing overheads involved in the data collection process in highly dynamic distributed environments.
- A straightforward application of existing algorithms for dynamic data management addresses the cost/accuracy trade-offs attempt to keep the database "reasonably" accurate; however, it ignores the arrival order and time constraints of user requests.

Determining an appropriate composition of the services for accuracy-aware scheduling and cost-aware database maintenance is challenging due to the following reasons: How the composition should work needs to be determined. Should the services operate independently with no interaction (resulting in poor performance) or should each service be extended to work in concert with the others? Each service becomes more complicated when extended and composed and there are no straightforward rules for how to adjust parameters for each service in order to achieve the overall best performance. It is also unclear exactly what information is needed to enable the composition to make

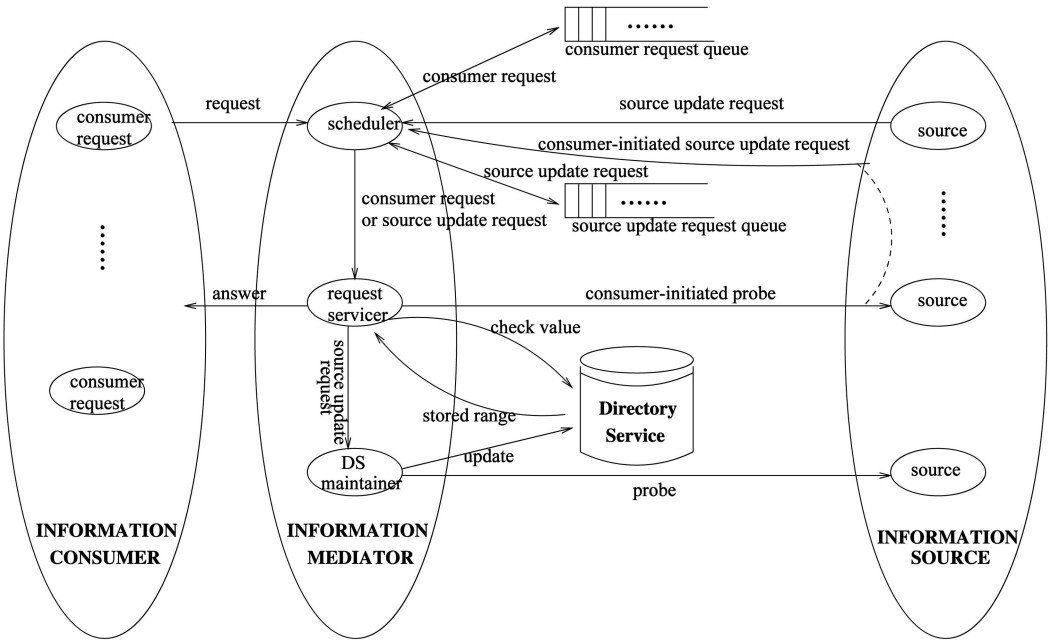


Fig. 1. A system architecture for real-time information collection.

the parameter adjustments. Furthermore, there are a number of dynamic factors affecting the data collection process: The network latency is unpredictable and variable in a best-effort network like the Internet, application workloads are highly dynamic, and information sources are frequently changing. Hence, adapting the system while balancing the composite trade-offs of timeliness, cost, and accuracy is not straightforward.

In order to enable a judicious composition of the scheduling and data management services for distributed data collection in highly dynamic environments, we propose a middleware framework for the real-time information collection process. Fig. 1 depicts the architectural components.

When a number of users request dynamic data at varying requirements under constantly changing system/network conditions, a mediation functionality is crucial to deliver the right data to the right user at the right time. Therefore, a key component of a real-time information collection architecture is an information mediator which connects information sources and consumers and serves as a crux of the information collection process where collection decisions are instrumented. Information sources communicate changes in source values to the mediator and information consumers forward their requests to the mediator. The mediator invokes suitable actions so that the DS maintains information at a suitable level of accuracy to satisfy the data quality and timeliness needs of consumers. Policies implemented in the mediator must appropriately represent the information in the DS, efficiently collect the information from sources, and effectively process requests from users.

The complex interrelationship between QoSSat and QoDSat illustrates the need for a scheduling mechanism that can work in concert with the DS maintenance component. Therefore, we frame the trade-off as two subproblems. We manipulate QoSSat by proposing an algorithm to schedule incoming source update requests and consumer requests, aiming to increase the number of consumer requests with their accuracy and deadline constraints met, thereby increasing QoSSat. This is done under the assumption that the DS is maintained with reasonable accuracy. Simultaneously, we focus on adjusting QoDSat by presenting an efficient DS maintenance algorithm that works in concert with the scheduling mechanism. The DS maintenance algorithm focuses on increasing QoDSat without increasing the management overhead.

Following this guideline, we design three subcomponents for the mediator. The *scheduler* aims to provide better QoSSat by prioritizing requests from both consumers and sources based on their urgency and popularity. The *DS maintainer* is responsible for data freshness (QoDSat), i.e., it maintains adequate DS accuracy to serve source/consumer requests while reducing collection overhead. The *request servicer* decides the specifics of processing each dispatched request and serves as a conduit between the scheduler and the DS maintainer. The request servicer determines whether the values stored in the DS are accurate enough for incoming requests and notifies the DS maintainer when accuracy violations occur. Based on this feedback, the DS maintainer adjusts its policy accordingly so that the DS is maintained at a reasonable accuracy level. In other words, via the request servicer, QoDSat is maintained by the DS maintainer to assist the scheduler in achieving better QoSSat. This is because QoDSat has a direct impact on the frequency of future source update requests and consumer requests, which affects the system load and schedulability, thus it indirectly has an impact on QoSSat. In Section 4, we develop customized algorithms to be implemented within each of these subcomponents to address the complexity to support distributed data collection application with multiple conflicting requirements: timeliness, accuracy, and cost-effectiveness.

4 Designing a QoSSat/QoDSat Balanced Information Mediator

A good scheduler and request servicer will ensure that each request will not wait for too long to get processed and that requests with deadlines will have the maximum possibility

TABLE 2
Absolute Deadline Assignments

request category	assigned absolute deadline ($ADL_{r_i(s,t)}$)
P-DL	$t + RDL_{r_i}$
P-NODL	$t + PER_{r_i}$
AP-DL	$max(t, ADL_{k-1}) + E_i/U_{AP}$
AP-NODL	$max(t, ADL_{k-1}) + E_i/U_{AP}$

of satisfying their deadlines. An effective DS maintenance module will keep the DS entries at the suitable level of accuracy so that requests can be served directly from the DS; this, in turn, will shorten the execution time. In the following sections, we present detailed techniques for scheduling (Section 4.1), request servicing (Section 4.2), and DS maintenance (Section 4.3).

4.1 Timeliness-Accuracy Balanced Scheduling (TABS)

Given that requests can arrive from both data sources and consumers, the request scheduler must very carefully balance the requirements of consumer requests and their deadlines (QoSSat) against the need to keep the DS entry up-to-date (QoDSat). If source updates are executed with higher priority, the system may be left with no time to meet the deadlines of consumer requests. On the other hand, if consumer requests are given preference, they might read stale data. Neither of these outcomes are desirable. Missing deadlines might mean missing opportunities, operating on stale data might mean making wrong decisions. The scheduling component must therefore determine the order in which the incoming source and consumer requests are handled so as to increase the possibility of meeting both accuracy and deadline constraints.

Solutions to address the conflict between user request timeliness and data freshness have been developed in the context of real-time databases (RTDB) [3], [4], [5]. There are some basic issues in mapping RTDB solutions to our scenario. In the RTDB context, consumer requests correspond to transactions (a series of read/write requests), which typically take a longer time to be processed. In our case, more fine-grained and timely interaction between the consumer and system is needed. Both source and consumer requests represent a single operation on a single data source. Furthermore, both source and consumer requests may specify timeliness requirements and a uniform mechanism for assigning a scheduling order to both types of requests is needed.

A suitable scheduling mechanism must address the following issues to obtain a balance between QoSSat and QoDSat: 1) Decide on an ordering of the incoming source update requests and 2) decide on a relative ordering of source update and consumer requests. We propose a Timeliness-Accuracy Balanced Scheduling (TABS) mechanism to balance timeliness and accuracy. TABS attempts to schedule consumer and source update requests to ensure that deadlines are met and source update requests are processed rapidly enough to maintain accuracy.

We classify all incoming source update and consumer requests into four categories shown in Table 2 based on periodicity and deadline, where the request r_i arrives at time t and k is the kth incoming aperiodic request. We motivate the utility of these categories via the following example: Consider a toxic chemical detection system that continuously monitors the density of a certain toxic chemical in an area. Under normal conditions, a *periodic nondeadline-based (P-NODL)* query is issued to the system; interested users may issue *aperiodic nondeadline-based (AP-NODL)* queries to check the density irregularly. When the density of the toxic chemical is above a certain threshold, *aperiodic deadline-based (AP-DL)* queries with explicit deadlines are issued so that a chemical threat can be quickly identified and false alarms can be avoided. Once a real threat is identified, *periodic deadline-based (P-DL)* queries may be issued to provide timely and accurate density level information to aid emergency response teams in mitigating the hazard.

Source update requests arrive at the system either periodically or sporadically. If multiple source update requests to the same source exist, the most recent update will be processed first (i.e., the most recent value is the candidate for a DS update) and the remaining values will be discarded. We then prioritize source update requests from different sources based on the popularity of the source, urgency, and deadline of the request.

For joint scheduling of source update requests and consumer requests, TABS uses the Total Bandwidth Server (TBS) algorithm [6] as a basis and the whole point of our algorithm is to assign an absolute deadline for each incoming request (see Table 2). We use the following assumptions and terminology:

- each periodic request r_i has a period PER_i and a constant worst-case execution time E_i;
- all periodic requests $r_i : i = 1, \ldots, n_p$ have deadlines specified either as RDL for P-DL or as its period PER for P-NODL;
- the arrival time of the lth periodic instance is given by $AR_i(l) = AR_i(l-1) + PER_i$;
- the absolute deadline of the lth periodic instance is given by $ADL_i(l) = AR_i(l) + RDL_i$ for P-DL or $ADL_i(l) = AR_i(l) + PER_i$ for P-NODL;
- all aperiodic deadline-based requests $ard_j : j = 1, \ldots, n_{apd}$ have relative deadlines RDL, so its original absolute deadline is

$$OADL_{ard_j} = AR_{ard_j} + RDL_{ard_j};$$

- all aperiodic nondeadline-based requests $arnd_k : k = 1, \ldots, n_{apnd}$ do not have user-specified deadlines;
- the worst case execution time of each aperiodic request E is known at its arrival time.

Assigning Absolute Deadlines for Aperiodic Requests. We define the processor utilization factor for periodic requests as $U_P = \sum_{i=1}^{n_P} \frac{E_i}{\min\{RDL_i, PER_i\}}$. U_{AP} refers to the capacity of the total bandwidth server (TB server), i.e., the total processor bandwidth allocated to aperiodic requests. To improve the response time of aperiodic requests, we assign a possible earlier deadline to each aperiodic request by applying the TBS algorithm [6]. Therefore, each time an aperiodic request enters the system, we optimistically assign it a deadline assuming that U_{AP} can be allocated to that request

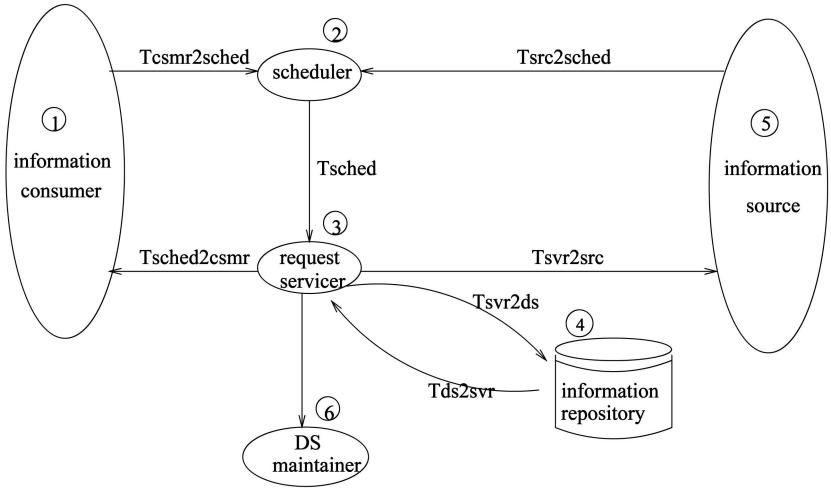


Fig. 2. Processing delays for request servicing.

immediately. When the kth aperiodic request arrives at time $t = AR_k$, it receives a deadline

$$ADL_k = max(AR_k, ADL_{k-1}) + \frac{E_k}{U_{AP}},$$

where E_k is the worst-case execution time of the request. By definition, $ADL_0 = 0$. If the request is aperiodic deadline-based, we compare the assigned absolute deadline ADL_k with its original absolute deadline $OADL_k$. If $ADL_k > OADL_k$, then the request is rejected. Otherwise, the request is inserted into the ready queue of the system and scheduled by EDF [7], as any other periodic instance or aperiodic request already present in the system. Intuitively, the assignment of the deadlines is such that, in each interval of time, the ratio allocated by EDF to the aperiodic requests never exceeds the U_{AP}; that is, the processor utilization of the aperiodic requests is at most U_{AP} [8].

Operational Flow of the TABS Algorithm. Based on the discussion above, for each incoming request, we set an absolute deadline by which the request must be completed (Table 2). Requests are ordered in their corresponding queues (source update and consumer request queue) by their absolute deadlines. When two requests have the same deadline, we calculate priority values to break the tie as the weighted sum of the popularity of the requested source (POP_s) and urgency of the request (UR_{r_i}). A request with a higher priority value will be assigned and dispatched earlier than a request with a lower priority value.

When a new request (either from a consumer or source) arrives, the scheduler inserts it into the corresponding queue so as to preserve deadline ordering and additionally preserve priority ordering for multiple requests with the same deadline. When a dispatched request finishes processing (at the request servicing module), the scheduler decides which request to process by comparing the deadlines in both source update request and consumer request queues: The request with the earliest absolute deadline or the highest priority is the next one to be dispatched to the request servicing module.

Lemma (TABS Schedulability). *Given a set of n_p periodic requests with processor utilization U_P and a TB server with processor utilization U_{AP}, the whole set of requests is schedulable if $U_P + U_{AP} \leq 1$.*

Proof. Suppose there is an overflow at time t. The overflow is preceded by a period of continuous utilization of the processor. Furthermore, from a certain point t' on, only instances of requests (periodic or aperiodic) ready at t' or later and having deadlines less than or equal to t are run. Let E be the total execution time demanded by these instances. Since there is an overflow at time t, we must have $t - t' < E$. Let E_{AP} be the total execution time required by aperiodic requests arrived at t or later and processed with deadlines less than or equal to t'; then, $E_{AP} \leq (t' - t)U_{AP}$ (Lemma 2 in [6]). We also know that

$$\begin{aligned} E &\leq \sum_{i=1}^{n_p} \left\lfloor \frac{t - t'}{\min\{RDL_i, PER_i\}} \right\rfloor E_i + E_{AP} \\ &\leq \sum_{i=1}^{n_p} \frac{t - t'}{\min\{RDL_i, PER_i\}} E_i + (t - t')U_{AP} \\ &\leq (t - t')(U_P + U_{AP}). \end{aligned}$$

It follows that $U_P + U_{AP} > 1$, a contradiction. □

4.2 Time-Sensitive Request Servicing

The request servicer accepts a source update or consumer request selected by the scheduler and determines the specifics of how individual requests will be processed in the system so as to satisfy the timeliness/accuracy/cost trade-off.

When the request servicer accepts a source update request, it forwards the request to the DS maintenance module that ultimately determines whether the DS should be updated. Handling consumer requests is more complicated than handling source update requests since consumer requests may have accuracy and timeliness requirements. When the request servicer accepts a consumer request, more careful analysis is needed to determine whether both accuracy and timeliness requirements can be met (ideally). If only the preferred requirement (as indicated by $Bias$) can be met, the request servicing module has to choose among several processing options. Fig. 2 illustrates the delays incurred during different operations.

We break down the request processing time into time spent in each step. Let $T_{csmr2sched}$ be the message transmission delay between the information consumer and scheduler, T_{sched} be the scheduler processing time, T_{svr2ds} be the

data request message transfer time from the request servicer to the DS, $T_{svr2src}$ be the message transfer time for the source probe from the request servicer to the source, $T_{svr2csmr}$ be the message transmission delay between the request servicer and the information consumer, T_{ds2svr} be the sum of the DS access time and the message transfer time from the DS to the request servicer, and $T_{src2sched}$ be the source update request transfer time from the source to the scheduler. Each consumer request follows the same path $1 \rightarrow 2 \rightarrow 3$ prior to reaching the request servicer. Subsequent to that, there are two possible paths/ways to serve the consumer request: 1) $PATH_A(3 \rightarrow 4 \rightarrow 3 \rightarrow 1)$: We obtain the desired value (range) from the DS; 2) $PATH_B(3 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1\&6)$: If accurate data is needed based on policy, we can probe the source to obtain the current exact value, return a response to the consumer, and concurrently issue a DS update (if necessary). Therefore,

$$T_{PATH_A} = T_{csmr2sched} + T_{sched} + T_{svr2ds} + T_{ds2svr} + T_{svr2csmr}, \text{ and}$$
$$T_{PATH_B} = T_{csmr2sched} + T_{sched} + T_{svr2ds} + T_{ds2svr} + T_{svr2src} + T_{src2sched} + T_{sched} + T_{svr2csmr}.$$

Each sublatency T_i is bounded by the best case and the worst case latency $[T_i(L), T_i(H)]$, so the total latency is bounded, i.e., if $T_i(L) \leq T_i \leq T_i(H)$ for any i, then $\Sigma T_i(L) \leq \Sigma T_i \leq \Sigma T_i(H)$. Given the estimated latency bounds and the timeliness/accuracy constraints of a consumer request *cr*, we can roughly determine if the deadline can be met as follows:

- If $RDL_{cr} \leq \Sigma T_i(L)$, then it is not possible to satisfy the timeliness bound.
- If $\Sigma T_i(L) < RDL_{cr} \leq \Sigma T_i(H)$, then it is uncertain if the deadline will be met or not. The consumer-specified Bias factor plays a role in determining the path of the request in this case.
- If $RDL_{cr} > \Sigma T_i(H)$, then the deadline can be met.

The general flow of handling consumer requests is as follows: A consumer request *cr* containing necessary parameters (as defined in Definition 2) is received by the request servicer. We decompose the processing of *cr* into a mandatory part followed by an optional part. As part of the mandatory processing, the current value is obtained from the DS. If the obtained value satisfies the accuracy constraints of *cr*, it is returned as the result. Otherwise, we must determine whether probing the source is necessary, i.e, whether the optional $PATH_B$ needs to be taken or not. Analysis of latency involved in $PATH_B$ indicates that $PATH_B$ will improve data accuracy but introduce a larger delay. Hence, providing the best service (accurate answer and timely response) for each incoming request may not lead to the best overall system performance. In contrast, by sacrificing the quality of some individual consumer requests, the overall QoSSat and QoDSat may be enhanced. In order to address this trade-off, we propose local and global policies that decide if $PATH_B$ will be taken.

Policies for Path Selection. Our objective is to support both timeliness and accuracy and tailor the result to the consumers' preference if the two constraints cannot be satisfied at the same time. We describe the policies in more detail below:

- Balanced Optima (BaO). This policy aims to meet the accuracy constraint while ensuring that the deadline is met, i.e., if the deadline will not be violated, $PATH_B$ will be taken if the precision from the DS is not good enough. More specifically, if the DS precision is not high enough, there are the following three cases to consider:
  - If $DL_{cr} \leq T_{PATH_B}(L)$, the value stored in the DS is returned as the answer.
  - If $T_{PATH_B}(L) < DL_{cr} \leq T_{PATH_B}(H)$, the deadline may be violated, whether or not $PATH_B$ is taken is dependent upon $Bias.cr$. If $Bias_{cr} = 0$ or $Bias_{cr} = 2$, i.e., the consumer has no preference or prefers to ensure accuracy, then $PATH_B$ will be taken and the obtained current value is returned as the answer; otherwise, if $Bias_{cr} == 1$, i.e., if the consumer prefers to ensure timeliness, then $PATH_B$ will not be taken and the value stored in the DS is returned as the answer.
  - If $DL_{cr} > T_{PATH_B}(H)$, then $PATH_B$ will be taken, i.e., the source will be probed and the obtained current value is returned as the answer.
- Biased Optima (BiO): From the overall system performance perspective, this policy focuses on ensuring either timeliness or accuracy and is oblivious of requests' preference.
  - Increase the deadline-meet ratio (BiO_{dl}): This policy increases the number of deadlines met by returning current value in the DS without probing the sources, thereby decreasing the processing time for each request.
  - Increase the accuracy-meet ratio (BiO_{ac}): If the DS precision is not high enough, this policy improves the accuracy of the result by probing the sources for the current exact values regardless of deadline violation.

4.3 Minimized Cost Directory Service Maintenance (MC)

The DS maintenance module is responsible for keeping the DS accurate enough so that most of the consumer requests can be served directly by consulting the DS without probing the sources. As a result, communication overhead and delay are both reduced. In addition, for each source, the DS maintains a range, which also affects the DS accuracy. Different applications are interested in the source values at different granularity, so it is desirable to sample the sources at a very high frequency initially so that the dynamics of the underlying sources can be captured. The goal of the DS maintenance algorithm is to find an efficient way to adjust collection parameters (sampling frequency, range size) so that desired information accuracy is maintained while minimizing the communication overhead, i.e., the algorithm should minimize the information collection cost while still maintaining reasonable information precision that can satisfy user requirements.

We propose the Minimized Cost DS Maintenance Algorithm (MC) by building upon the approach presented in TRAPP [1]. The update of DS stored ranges may be a result of a value-initiated update (source value falls outside predetermined threshold) or a consumer-initiated update (to

improve result accuracy for the consumer). They analyzed all the cost factors in the whole collection process and justified that the cost is minimized if the ratio of frequency of consumer-initiated update to the frequency of value-initiated update is maintained to be a constant. In their approach, consumer-initiated updates always result in range reduction, while value-initiated updates always lead to range relaxation. However, we observe that range relaxation for all value-initiated updates does not truly reflect the value changes. The fact that a source value falls outside the current range does not always imply that the range is too small and needs to be increased; it can also occur when the source value gradually shifts towards a new set of values.

In our approach to value-initiated range adjustment, the general idea is to compare the approximate trend of source value changes using the following steps: Whether the range should be expanded or tightened is dependent upon 1) the absolute slope of the trend of source value changes during the current monitoring period and 2) the difference between the current slope and previous one. To find the slope of the trend of source value changes, we use a Least Squares curve fitting method to find the most suitable line to approximate source value changes.[1] The slope of the obtained line represents the rate of change of source values. If the values in a sliding window w of size N are $V_1, V_2 \ldots, V_N$, then the slope $m_w = \frac{n\Sigma xy - (\Sigma x)(\Sigma y)}{n\Sigma x^2 - (\Sigma x)^2}$, where $\Sigma \ldots$ stands for $\Sigma_{i=1}^{N} \ldots_i$. We compare m_w with a specified threshold TH_m to decide whether the source change is significant enough. The comparison between m_w and m_{w-1} is made against a predefined ε. When the current change exceeds the threshold, we adjust the range as follows:

- the range size is increased to accommodate more changes if the current change m_w is bigger than the previous change m_{w-1},
- the range size remains the same if m_w is similar to m_{w-1}, and
- the range size is decreased if the change is smaller than the previous one.

When the current change is not significant enough, the range size is decreased if the previous change m_{w-1} is larger than the current change m_w; otherwise, the range size remains the same. When the range is tightened, $R_{new} = R_{old}/(1+\alpha)$; when the range is relaxed, $R_{new} = R_{old} \cdot (1+\alpha)$. No matter how the range is changed, we let the median value V_{med} in the current sliding window fall at the center of the range, i.e., $U = V_{med} + \frac{R_{new}}{2}$ and $L = V_{med} - \frac{R_{new}}{2}$.

Integrating TABS and MC via Request Servicing. The integration of our proposed algorithms for request scheduling (TABS), servicing, and DS maintenance (MC) maintains a good balance between QoSSat, QoDSat, and Cost. TABS provides a joint scheduling of both source update requests and consumer requests, thus ensuring that one type of request is not delayed because extensive resources are allocated to the other type. MC keeps track of the changes in the source values and system conditions, adjusts DS representations to reflect these changes, and makes sure that the DS closely reflects the real world without incurring too much maintenance overhead. The request servicer serves as a conduit between the scheduler and the DS maintainer. The well-maintained DS provides TABS with convincing information to make its scheduling decisions. TABS, in turn, provides MC with valuable feedback about whether each incoming request meets its requirement. These information exchanges are done via request servicer. Overall, the combination of our proposed approaches performs very well under varying system conditions, which will be shown in the following section.

1. With heavy workload, computational resources may potentially become a bottleneck and computation delay may dominate the total latency of the collection process [9]. However, we currently only address the case where the mediator has sufficient computation resources and is not overloaded. This assumption will be relaxed in future work.

4.4 Extensions to Support Distributed Mediators

Our discussion so far has been assuming that the mediator is centralized. Several questions need to be answered in order to support distributed mediation:

1. How can we ensure load balancing among distributed mediators?
2. Where and how can we place/replicate data among multiple directory services?
3. How can we maintain a consistent view if the data is replicated?
4. Where can we retrieve the requested data to ensure timeliness and accuracy?

We now extend the architecture to support distributed mediation. The system consists of n mediators and n fully replicated directory services. In order to balance the load among all the mediators, an incoming request is assigned to the least loaded mediator upon arrival. For a consumer request, it retrieves the value from the assigned directory service; if the stored value does not meet the accuracy constraint, the information source is contacted for the most recent value. While designing optimal consistency management policies is beyond the scope of this paper, we perform some preliminary studies and explore the cost and benefits of maintaining 1) strongly consistent, 2) inconsistent, and 3) weakly consistent directory services. For strongly consistent directory services, an update to any directory service will cause the same updates to the other directory services. For inconsistent directory services, an update to a directory service does not propagate to the other directory services. For weakly consistent directory services, an update to a specific directory service is not propagated to other directory services immediately. However, a consistency check is performed periodically so that the directory services can be updated with the most recent values. A full-fledged treatment of distributed mediation for real-time information collection is part of our ongoing work, which can benefit from existing research on distributed databases, efficient replicated services, and distributed network management.

5 Performance Evaluation

The objective of the simulation is to study in detail the performance of the system by comparing our algorithms with existing algorithms proposed by other previous work. In addition to QoSSat, QoDSat, and Cost, we also use EoS (Efficiency of System) $EoS = (QoSSat \cdot QoDSat)/Cost$ as an overall metric.

We compare TABS with the following scheduling approaches [3] proposed in the context of real-time databases:

1. *Source update request First (SF)*: This approach applies the source update request when it arrives at the system, i.e., it gives all source update requests higher priority than all consumer requests.
2. *Consumer request First (CF)*: This approach applies source request updates only when no consumer requests are waiting.
3. *Split Updates (SU)*: This approach is a compromise between CF and SF and classifies data objects as being popular and unpopular. Source update requests to popular data will be applied on arrival and updates to less popular data will be applied when no consumer requests are waiting.
4. *On-Demand source request updates (OD)*: This approach is an extension to the CF (consumer first) policy where consumer requests are normally given precedence over source update requests. However, when a consumer request encounters a stale object (i.e., an object for which there exists a source update that has not yet been applied), the corresponding source update is given precedence.

MC is compared against the following relevant DS maintenance approaches:

1. *Instantaneous Snapshot-Based Information Collection (SS)*: In this policy [10], information about the desired parameters is based on an absolute value obtained from a periodic snapshot.
2. *Static Interval-Based Information Collection (SI)*: In this policy [11], a fixed interval B is defined to partition the capacity of the collected information into a fixed number (say, n) of equal size classes:

$$(0, B), (B, 2B), (2B, 3B), \ldots, ((n-2)B, (n-1)B).$$

   The classes are represented by corresponding indices $0, 1, 2, \ldots (n-1)$. A probe is initiated at each sampling interval. If the obtained value is out of the range indicated by the current index, the DS is updated with another index; otherwise, no update is needed.
3. *Dynamic Range-Based Information Collection*: In this policy, the DS holds the monitored parameter using a range with an upper bound U and a lower bound L; the range may be modified dynamically based on the sampled information. In our prior studies, we have shown that, among a family of dynamic range-based policies, a simpler throttle-based (TR) [12] approach performs well, where ranges are increased or decreased exponentially using a prespecified throttle factor.

With six scheduling algorithms, three request servicing algorithms, and four DS maintenance algorithms, we have 72 combinations of the policies. Our performance study consists of the following specific experiments:

1. Evaluation of all the possible policy combination in terms of the overall EoS.
2. Evaluation of system heterogeneity: In a general architecture, source capabilities can vary significantly. For instance, some sources can only respond to a request, while others have certain computing capability. Also, different applications exhibit varying deadline requirements from the information collection process. We study the system heterogeneity under these variations.
3. Evaluation of relative merit of adding intelligence into each module of the mediator and the benefit of intelligent mediator policies as the system scales.
4. Evaluation of the scalability of distributed mediators.

While we studied all the combinations, we only illustrate the key results here.

The Simulation Parameter Settings. We next describe how information sources and the two types of incoming requests (source update requests and consumer requests) are modeled. There are $n = 200$ sources in the system. Each source holds one exact numeric value, the local cache in the request servicing module holds $k \leq n$ interval approximations to exact source values, and the DS holds all the interval approximations. Source values V_s are picked randomly and uniformly in the range [-150, 150]; source values are changed periodically (initially set to be every 100 milliseconds): Some of the sources change their values very slightly from ± 0.5 to ± 1.5, while others change more dramatically from ± 5 to ± 15.

For each source, a source update request is sent out regularly and the period is uniformly distributed in the range $(100ms, 50s)$. The arrivals of aperiodic source update requests are dependent upon their source value changes: We randomly pick sources that send out urgent source update requests when their values reach certain thresholds. For requests that have timeliness requirements, deadlines are uniformly distributed in the range $(500ms, 1min)$. All sources have an equal probability of generating a source update request.

One periodic consumer request is issued for each source and its period is uniformly distributed in the range $(100ms, 50s)$. An aperiodic consumer request arrival is modeled as a Poisson process with arrival rate $10/sec$ and with interarrival times being exponentially distributed. Deadlines associated with consumer requests are uniformly distributed in the range $[500ms, 1min]$. It is randomly decided whether the aperiodic consumer request is deadline or non-deadline-based. Furthermore, each source has an equal probability of being requested by consumer requests. The urgency of each consumer request is randomly chosen to be 0, 1, or 2. Each request is accompanied by an accuracy constraint $PREC_{cr}$ specifying the maximum acceptable width of the result. The accuracy constraint is generated randomly from the range [0, 40].

Experimental Results. Fig. 3 shows the system efficiency classified by the six scheduling algorithms: TABS, FCFS, CF, SF, SU, and OD. Overall, we observe that the two combinations of policies ($\mathrm{TABS} + BiO_{ac} + \mathrm{MC}$ and $\mathrm{TABS} + BiO_{ac} + \mathrm{SI}$) result in highest EoS. Both of them use TABS as the scheduling policy. This is because TABS keeps a good balance between source update requests and consumer requests, thus rendering reasonably good QoSSat and QoDSat; in addition, both combinations use BiO_{ac} as the request servicing policy. This is because, in policy BiO_{ac}, sources are probed for each consumer request to improve QoDSat regardless of how well the DS is maintained.

In addition, Fig. 3 indicates the lowest EoS of policy combinations using BiO_{dl} as the request servicing policy. To

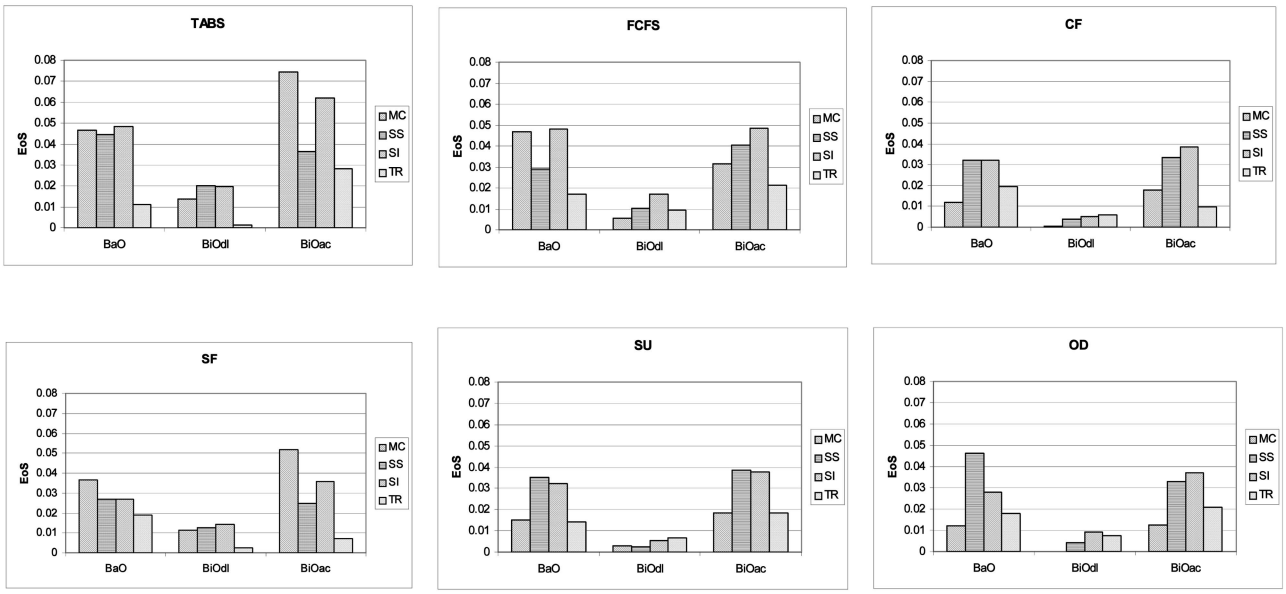
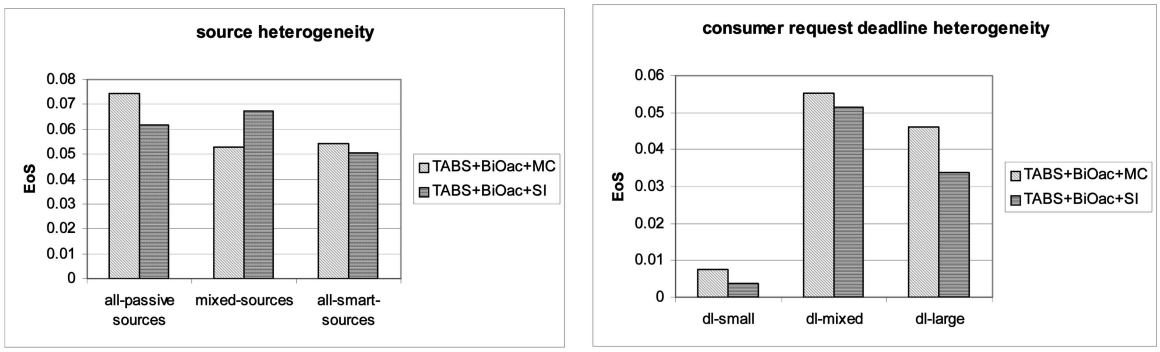


Fig. 3. Overall performance of mediator policies: This figure illustrates the performance of the six request scheduling policies under different combinations of request servicing policies and DS maintenance policies.

further identify the reasons, we analyzed the three factors (QoSSat, QoDSat, and Cost) of EoS for the combinations of TABS, MC, and the three different consumer request servicing polices (BaO, BiO_{dl}, and BiO_{ac}). Due to space limitation, we omit the figure here. BiO_{dl} returns the values in the DS for all the consumer requests, hoping to shorten each request processing time, thereby increasing the deadline meet ratio. However, the other factor that affects the system performance is the accuracy meet ratio and it is greatly decreased under BiO_{dl}. In addition, we also observe that the QoDSat of using BiO_{dl} is lower than using BaO and BiO_{ac}. BiO_{dl} does not invoke source probings, which means that no consumer requests trigger any DS updates, thus limiting the accuracy of the DS representation and lower QoD. Therefore, even though BiO_{dl} seemingly saves some overhead by not probing sources, the overall system efficiency is still very low.

System Heterogeneity Evaluation. In our basic performance evaluation, we assume all the sources are *passive*, i.e., they can only respond to probes. In the real world, some sources may have the capability of reporting significant changes in their own values. This requires basic processing and memory capabilities at the source; we designate such sources to be *smart sources*. This additional intelligence relieves the mediator from the burden of probing; however, resources may be wasted when no consumers are interested in the collected values. We discuss how the two best combinations of policies adapt to source heterogeneity for the following cases: 1) *all-passive-sources*: All of the sources are passive. 2) *mixed-sources*: A mixed configuration with smart and passive sources (50 percent of each in the base case). 3) *all-smart-sources*: All of the sources are smart. From Fig. 4a, we observe that, with an increasing number of smart sources, the performance of $(\mathrm{TABS} + BiO_{ac} + \mathrm{MC})$ decreases. The reason behind the counterintuitive results may be that, when sources are intelligent enough to participate in decision making, the Cost increases, since they are typically distributed in the network. This also explains why the performance of $(\mathrm{TABS} + BiO_{ac} + \mathrm{SI})$ is the best when there are mixed sources in the system.

Fig. 4b demonstrates system performance as the deadlines of the consumer requests vary. We classify three ranges of possible deadlines: 1) *dl-small*: The deadlines are



(a) (b)

Fig. 4. Effect of source/request heterogeneity on the performance of mediator policies.

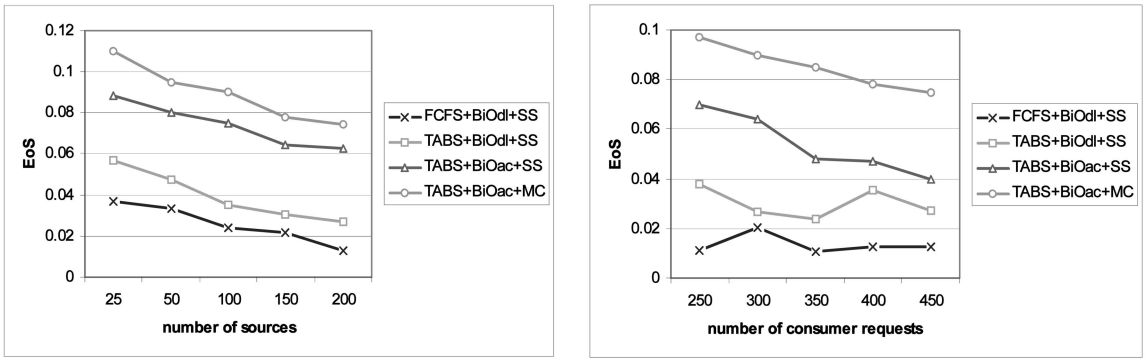


Fig. 5. Scalability of the composite policies.

uniformly distributed from 500 ms to 1 second. 2) *dl-mixed*: The deadlines are uniformly distributed from 500 ms to 1 minute. 3) *dl-large*: The deadlines are uniformly distributed from 50 second to 1 minute. Fig. 4b illustrates that, when all the deadlines are very small, none of the combinations exhibit very high EoS, since the deadlines are so small that even just getting data from the DS cannot meet the deadlines. As the deadlines get bigger, EoS increases greatly since meeting deadlines becomes very easy. It is interesting to note that, when the deadlines get even bigger, the EoS of both policy combinations actually decreases. We believe this is because, when meeting deadlines is not a concern anymore, most requests will be served by obtaining most recent values from the sources, which incurs significant overhead.

Scalability Evaluation of Composite Policies. We show the benefits of adding intelligence to each of the three components of the mediator. FCFS, BiO_{dl}, and SS are the simplest policies, respectively, for scheduling, request servicing, and DS maintenance. We gradually enhance the intelligence of the system by enhancing the scheduler $(\mathrm{TABS} + BiO_{dl} + \mathrm{SS})$, followed by the request servicer $(\mathrm{TABS} + BiO_{ac} + \mathrm{SS})$, and, finally, arriving at the most intelligent set of policies by enhancing the DS maintainer $(\mathrm{TABS} + BiO_{ac} + \mathrm{MC})$. Fig. 5a shows the benefits of adding more intelligence to each component gradually and also the changes of the benefits as the number of sources increases. As expected, the EOS increases as more intelligence is added to each component. Merely adding intelligence to the scheduler (i.e., replacing FCFS with TABS) shows marginal benefit, but combining this with an intelligent request servicer (i.e., replacing BiO_{dl} with BiO_{ac}) improved the EOS significantly. This implies the necessity of combining the scheduler and the request servicer to ensure both fairness and efficiency of request processing. In addition, adding intelligence to the DS maintainer (i.e., replacing SS with MC) decreases the overhead involved in maintaining the DS, thus further increasing the EoS. Note, however, that as the number of the sources in the system increases, the EoS decreases since the system is busy handling more source update requests. We observe that the enhanced mechanisms remain useful as the number of sources increases. Fig. 5b shows the change of the benefits as the number of consumer requests increases. In general, it shows the similar trend as the number of sources increases.

Scalability Evaluation of Distributed Mediation. We now empirically study the cost and benefits of the three consistency management policies discussed above. When *strong consistency* is maintained among distributed directory services, the increase in the number of consumer requests leads to increasing EoS (Fig. 6a). However, when the number of mediators/directory services reaches a threshold (five in our case), EoS decreases. This observation shows that, while distributing mediation tasks among multiple mediators does relieve resource contention, maintaining strong consistency among fully replicated directory services incurs significant

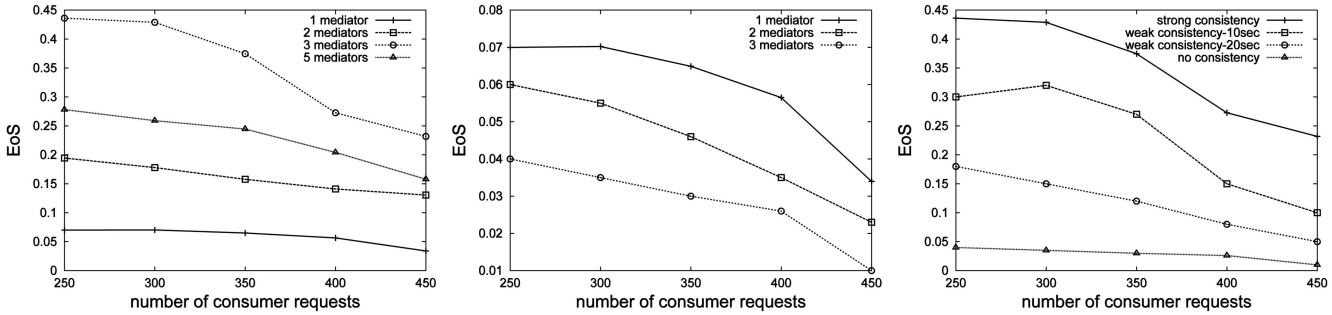


Fig. 6. Impact of distributed mediators on system performance of policy combination $[\mathrm{TABS} + BiO_{ac} + \mathrm{MC}]$. (a) Strongly consistent mediators. (b) Inconsistent mediators. (c) Weakly consistent mediators.

overhead. Therefore, increasing the distribution beyond a certain point, in fact, is detrimental to the overall system performance. When *no consistency* is maintained, any updates to the directory service are local and do not propagate to other directory services. Fig. 6b indicates that the increase of the number of mediators leads to decreasing EoS. This is because any updates to the directory services are localized, and QoDSat decreases, thus leading to lower QoDSat, which further results in lower EoS when *weak consistency* is maintained among n ($n = 3$ here) distributed directory services. We tested the case when the update period is 10 seconds and 20 seconds. The policies where strong or no consistency is maintained as described above are special cases of this policy. Fig. 6c indicates that, as the update period increases, the directory services are left with more divergent values and QoDSat decreases; hence, EoS decreases.

Performance Summary. Our performance studies indicate that the policies combination of $(\mathrm{TABS} + BiO_{ac} + \mathrm{MC})$ exhibits higher system efficiency as compared to the simple police compositions of $(\mathrm{FCFS} + BiO_{dl} + \mathrm{SS})$. This illustrates the advantage of enhancing the three main components of the system with intelligent strategies. We also find that, as the system scales in terms of the number of sources or the number of consumer requests, this combination continues to perform well. In addition, the two best policy combinations under the generic scenario are very robust to the source heterogeneity and consumer request deadline variation. The studies on the scalability of distributed mediators indicate that merely increasing the number of mediators is not sufficient to improve system scalability. There exists an inherent trade-off between average request response time and the cost involved in maintaining consistency.

6 Related Work

In this section, we compare our work to related research in temporal and real-time databases and dynamic data management.

Temporal and Real-Time Databases. After researchers identified the need to incorporate both temporal and real-time capabilities [13], [14] to ensure temporal data consistency while providing time-constrained transaction processing, several projects (such as STRIP [3], ARCS [4], and QMF [5]) address the trade-off between transaction timeliness and data freshness. Our work differs from the aforementioned prior research in several ways. First, a regular transaction is a sequence of database operations, whereas the incoming requests in our case (consumer or source update requests) are special transactions with only one single operation. Real-time transaction processing techniques focus on the optimized execution of a group of operations to meet transaction deadlines; in our case, the deadlines are on the fine-grained operations themselves. This implies that we must explore trade-offs between fine-grained operations, and traditional real-time transaction processing techniques will not suffice. Second, data is represented approximately in our system in order to reduce communication overhead for maintaining the repository. The approximation also implies that source update requests can have a direct impact on the data accuracy, which can indirectly affect timeliness. Our objective is to maintain a separation of concerns and isolate tasks within the real-time path flow (scheduling, request servicing) from tasks that maintain accuracy and minimize cost (DS maintenance). At the same time, we would like to allow adequate interaction between these components to balance the cost/accuracy/time trade-offs.

Dynamic Data Management. Approximate data caching approaches [15], [1], [12], [16], [17], [18] have been proposed to address the trade-off between the accuracy of cached data and the overhead involved in maintaining the cache. In our work, we address time constraints in addition to precision requirements imposed on user queries; this further complicates the DS maintenance problem. In addition, our work considers system performance in terms of overall deadline satisfaction ratio, rather than for each individual query. Therefore, a scheduling mechanism is needed.

Data streaming has been a vibrant research field in recent years for both traditional Internet environments (e.g., OpenCQ [19], Niagara [20], Telegraph [21], STREAM [22], and Aurora [23]) and emerging ad hoc sensor networks (e.g., COUGAR [24] and Quasar [25]). Researchers in this area have addressed a whole gamut of issues, such as system architectures, concepts/semantics of continuous queries, QoS specifications for fresh information delivery, system scalability, etc. Our work targets a similar context where data is fast changing and update intensive, but we provide support for collecting and accessing the data in real-time. In addition, we take advantage of an application's tolerance toward data imprecision and address the three factors—request timeliness, data accuracy, and maintenance overhead.

The real-time data collection problem bears some similarity to dynamic Web content caching [26], [27], [28], [29], [30] at a higher level. Since changes to underlying data might affect more than one Web page, the key issues in dynamic Web content caching are to determine what dynamic pages should be cached and when a cached page has become obsolete. Although both timeliness and accuracy matter here, Web requests do not have explicit time constraints associated, while the data retrieval requests in our scenario may arrive into the system with user-specified deadlines. In addition, in the dynamic Web content caching problem, it is the Web pages that are being cached; on the other hand, we store the dynamic data (more specifically, numerical data such as stock prices, network measurement data, etc.). We can therefore use range-based approximation to represent the dynamic data and then exploit applications' tolerance toward data imprecision to decrease data collection overhead.

7 Conclusions

A key concern of our work is to exploit the accuracy and latency margins to ensure that most applications receive information at the desired levels of accuracy and timeliness while minimizing the consumption of resources (storage, network bandwidth, etc.) in distributed environments. In this paper, we have shown, using the proposed real-time information collection architecture, how an application's tolerance of information imprecision can be taken into account to improve system timeliness. We have also indicated that the trade-off between timeliness and accuracy can be addressed by scheduling requests from both sources and consumers in a uniform manner (i.e., not favoring either type). At the same time, we observe that a well-maintained DS (in concert with the scheduling algorithm) is essential for balanced QoSSat and QoDSat under

different system load and input characteristics. The eventual goal of our work is to develop effective tools for context-aware applications in highly dynamic and distributed environments. Middleware techniques for achieving the competing goals of timeliness, accuracy, and cost-effectiveness, such as those described in this paper, are key to delivering the right information to the right user at the right time in pervasive computing environments. This paper forms the basis for future research in the direction of bringing together real-time systems and data management techniques through composable middleware systems.

REFERENCES

[1] C. Olston, B.T. Loo, and J. Widom, "Adaptive Precision Setting for Cached Approximate Values," *Proc. ACM SIGMOD,* 2001.

[2] A. Labrinidis and N. Roussopoulos, "Update Propagation Strategies for Improving the Quality of Data on the Web," *Proc. Int'l Conf. Very Large Data Bases (VLDB '01),* 2001.

[3] B. Adelberg, H. Garcia-Molina, and B. Kao, "Applying Update Streams in a Soft Real-Time Database System," *Proc. ACM SIGMOD,* 1995.

[4] A. Datta and I. Viguier, "Providing Real-Time Response, State Recency and Temporal Consistency in Databases for Rapidly Changing Environments," *Information Systems,* vol. 22, no. 4, 1997.

[5] K.D. Kang, S.H. Son, J.A. Stankovic, and T.F. Abdelzaher, "A QoS-Sensitive Approach for Miss Ratio and Freshness Guarantees in Real-Time Databases," *Proc. 14th Euromicro Conf. Real-Time Systems,* 2002.

[6] M. Spuri and G. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Real-Time Systems,* vol. 10, no. 2, 1996.

[7] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, 1973.

[8] M. Spuri, G. Buttazzo, and F. Sensini, "Robust Aperiodic Scheduling under Dynamic Priority Systems," *Proc. IEEE Real-Time Systems Symp. (RTSS '95),* 1995.

[9] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, "A Scalable Solution to the Multi-Resource QoS Problem," *Proc. IEEE Real-Time Systems Symp. (RTSS '99),* 1999.

[10] M. Stemm, R. Katz, and S. Seshan, "A Network Measurement Architecture for Adaptive Applications," *Proc. INFOCOM,* 2000.

[11] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi, "Quality of Service Based Routing: A Performance Perspective," *Proc. ACM SIGCOMM,* 1998.

[12] Q. Han and N. Venkatasubramanian, "AutoSeC: An Integrated Middleware Framework for Dynamic Service Brokering," *IEEE Distributed Systems Online,* vol. 2, no. 7, 2001.

[13] G. Ozsoyoglu and R.T. Snodgrass, "Temporal and Real-Time Databases: A Survey," *IEEE Trans. Knowledge Data Eng.,* vol. 7, no. 4, Aug./Sept. 1995.

[14] K. Ramamritham, R. Sivasankaran, J. Stankovic, D.T. Towsley, and M. Xiong, "Integrating Temporal, Real-Time and Active Databases," *ACM SIGMOD RECORD,* vol. 25, no. 1, 1996.

[15] Y. Huang, R. Sloan, and O. Wolfson, "Divergence Caching in Client-Server Architectures," *Proc. IEEE Int'l Conf. Parallel and Distributed Information Systems (PDIS '94),* 1994.

[16] C. Olston and J. Widom, "Best-Effort Cache Synchronization with Source Cooperation," *Proc. ACM SIGMOD,* 2002.

[17] C. Olston and J. Widom, "Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data," *Proc. Int'l Conf. Very Large Data Bases (VLDB '00),* 2000.

[18] S. Shah, K. Ramamritham, and P. Shenoy, "Maintaining Coherency of Dynamic Data in Cooperating Repositories," *Proc. Int'l Conf. Very Large Data Bases (VLDB '02),* 2002.

[19] L. Liu, C. Pu, and W. Tang, "Continual Queries for Internet Scale Event-Driven Information Delivery," *IEEE Trans. Knowledge and Data Eng.,* vol. 11, no. 4, pp. 610-628, July/Aug. 1999.

[20] J. Chen, D.J. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," *Proc. ACM SIGMOD,* 2000.

[21] S.R. Madden and M.J. Franklin, "Fjording the Stream: An Architecture for Queries over Streaming Sensor Data," *Proc. IEEE Int'l Conf. Data Eng. (ICDE '02),* 2002.

[22] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," *Proc. ACM Symp. Principles of Database Systems (PODS '02),* 2002.

[23] D. Carney, U. Cetintemel, M. Cherniack, C.C.S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring Streams: A New Class of Data Management Applications," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02),* 2002.

[24] P. Bonnet, J.E. Gehrke, and P. Seshadri, "Towards Sensor Database Systems," *Proc. IEEE Int'l Conf. Mobile Data Management (MDM '01),* 2001.

[25] I. Lazaridis, Q. Han, X. Yu, S. Mehrotra, N. Venkatasubramanian, D. Kalashnikov, and W. Yang, "Quasar: Quality Aware Sensing Architecture," *ACM SIGMOD Record,* vol. 33, no. 1, pp. 26-31, Mar. 2004.

[26] A. Iyengar and J. Challenger, "Improving Web Server Performance by Caching Dynamic Data," *Proc. Usenix Symp. Internet Technologies (USEITS '97),* 1997.

[27] J. Wang, "A Survey of Web Caching Schemes for the Internet," *ACM SIGCOMM Computer Comm. Rev.,* vol. 29, no. 5, 1999.

[28] H. Zhu and T. Yang, "Class Based Cache Management for Dynamic Web Content," *Proc. INFOCOM,* 2001.

[29] J. Challenger, P. Dantzig, A. Iyengar, M. Squilante, and L. Zhang, "Efficiently Serving Dynamic Data at Highly Accessed Web Sites," *IEEE/ACM Trans. Networking,* vol. 12, no. 2, 2004.

[30] L. Ramaswamy, L. Liu, A. Iyengar, and H. Venkateswaran, "Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '05),* 2005.

Qi Han received the MS degree in computer science from the Huazhong University of Science and Technology, China, and the PhD degree in computer science from the University of California, Irvine. She is an assistant professor in the Department of Mathematical and Computer Sciences, Colorado School of Mines. Her research interests include distributed systems, middleware, mobile and pervasive computing, systems support for sensor applications, and dynamic data management. She is specifically interested in developing adaptive middleware techniques for next generation distributed systems. She is a member of the IEEE and ACM.

Nalini Venkatasubramanian received the MS and PhD degrees in computer science from the University of Illinois, Urbana-Champaign. She is an associate professor in the School of Information and Computer Science, University of California, Irvine. Her research interests include distributed and parallel systems, middleware, mobile environments, multimedia systems/applications, and formal reasoning of distributed systems. She is specifically interested in developing safe and flexible middleware technology for highly dynamic environments. She was a member of the technical staff at Hewlett-Packard Laboratories in Palo Alto, California, for several years, where she worked on large scale distributed systems and interactive multimedia applications. She has also worked on various database management systems and on programming languages/compilers for high-performance machines. She is a member of the IEEE and ACM.