Addressing Timeliness/Accuracy/Cost Tradeoffs in Information Collection for Dynamic Environments

Qi Han Nalini Venkatasubramanian
Department of Computer Science, University of California-Irvine
{qhan,nalini}@ics.uci.edu

Abstract

In this paper, we focus on addressing the tradeoffs between timeliness, accuracy and cost for applications requiring real-time information collection in distributed real-time environments. In this scenario, information consumers require data from information sources at varying levels of accuracy and timeliness. To accommodate the diverse characteristics of information sources and varying requirements from information consumers, we use an information mediator to coordinate and facilitate communication between information sources and consumers. We develop algorithms for real-time request scheduling and directory service maintenance and compare our techniques with several other proposed strategies. Our studies indicate that the judicious composition of our proposed intelligent policies can improve the overall efficiency of the system. Furthermore, the proposed policies perform very well as the system scales in the number of information sources and consumer requests.

1. Introduction

Recent developments in mobile computing, communications and embedded systems are likely to enable the deployment of large scale ubiquitous computing environments. We expect that real-time information services will gain importance with the emergence of pervasive computing environments. 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. Firstly, 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. Secondly, users requiring access to this dynamically changing data present variable user requirements in terms of 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. 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.

The key component of a real-time information collection architecture is an information mediator. Information sources communicate changes in source values to the mediator and information consumers forward their requests to the mediator. Given the data intensive nature of the system, the real time information collection architecture includes an information repository, i.e. directory service (DS) that reflects the intensive data changes as closely as possible or necessary. Maintaining the DS is not a trivial task, as the data changes constantly and data updates arrive frequently. 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 tradeoff 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.

In this paper, we develop strategies to address the timeliness/accuracy/cost tradeoff in real-time information collection. We propose a middleware framework for the real-time information collection process and design algorithms for real-time scheduling and DS maintenance. Through extensive performance studies, we determine a judicious composition of mediation policies that address the real-time information collection tradeoffs under varying conditions.

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

2. A Middleware Framework for Real-time Information Collection

Figure 1 depicts the architectural components of our real-time information collection framework. The framework is designed to operate in highly dynamic environments and consists of five components:

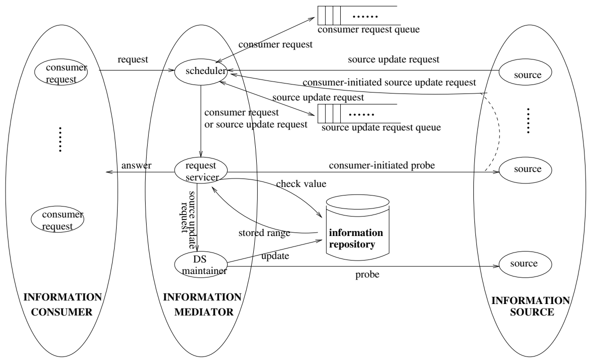


Figure 1: **A system architecture for real-time information collection**

• *Information Source:* This is the managed entity, such as the server, link, sensor or mobile/fixed host. The information sources can be programmed to send updates periodically, or send updates when something abnormal occurs.
• *Information Repository:* Information repository is used to hold information about sources that are of interest to users. 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.). In our system, we use the directory service (DS) as the information repository.
• *Information Consumer:* This module consumes data collected from the information sources (stored in the directory service) for application and system level tasks. 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.
• *Request Queues*: Two queues (consumer request queue and source update request queue) are maintained to buffer consumer requests and source update requests. The requests are ordered by their assigned priorities such that the first element of the queue has the highest priority.
• *Information Mediator:* This module serves as the decision point of the information collection. It processes reports from sources and notifications from consumers and invokes suitable actions so that the directory service maintains information at a suitable level of accuracy to satisfy the data quality and timeliness needs of consumers. Given that the information is the basis of the system, policies implemented in the mediator components must appropriately represent the information in the DS, efficiently collect the information from sources and effectively process requests from users.

The mediator consists of three sub-components (scheduler, request servicer and DS maintainer) that perform the following tasks: serve requests from consumers (i.e., assign priorities to queries and dispatch the requests); serve requests from sources (i.e., prioritize data based on their urgency and popularity, and dispatch them accordingly); maintain adequate DS accuracy to serve consumer requests and source requests while reduing collection overhead.

3. Problem Formulation

In this section, we describe data and request models for the highly dynamic environments, characterize the performance metrics QoS (Quality of Service) and QoD (Quality of Data) to be used in evaluating the system, and provide a formal definition of the real-time information collection problem. Currently, we assume the presence of a single centralized DS. We also assume that users only ask for current data, not historical data at a specified time instant.

We assume that the system consists of a number of data sources. The data obtained is reported by sources distributed in the environment being modeled and collectively describes the state of the system. Examples include link utilization from the network management domain, stock and commodity prices from the financial trading domain, temperatures and pressures from a chemical process control domain. We specifically consider systems where state information changes rapidly generating a large amount 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 queried (read only) from the DS by consumers. Oftentimes data from certain sources are 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.

To provide more flexibility to applications, we provide two parameters (urgency and 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 deadline of a request specifies the specific time by which the request must be completed. Application may specify one or both of these parameters (i.e., urgency and deadline); when both are present, the 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:

$$RDL_r = \begin{cases} 0 & \text{no deadline} \\ d & r \text{ is to be finished in } d \text{ time units} \end{cases}$$

The precision of a range is defined as the reciprocal of the range size,i.e., $PREC(L, U) = \frac{1}{U-L}$. Therefore, a zero-width range contains the exact value and its precision is infinite, while an infinite-width range has no information about the exact value and its precision is zero.

Each consumer request is accompanied by a time constraint, and also a precision constraint specifying the maximum acceptable width of the result. When joint constraint satisfaction is not possible, we make use of a user specified 'Bias' factor that indicates the preferences of consumers.

$$Bias_{cr} = \begin{cases} 0 & \text{no preference} \\ 1 & \text{timeliness weighs more than accuracy} \\ 2 & \text{accuracy weighs more than timeliness} \end{cases}$$

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

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, i.e., if current range size is R, then $U = V + R/2$ and $L = V - R/2$. The DS maintenance component is responsible for adjusting and storing the ranges.

Consumer requests are read-only requests submitted to the system to query current values of data sources.

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 $PREC$, accuracy or deadline bias $Bias$, $cr :=< s, t, PER, UR, RDL, PREC, Bias >$.*

3.1. Characterizing QoS and QoD

To satisfy the precision constraints, the precision of answer A must be adequate, i.e., $PREC_A \geq PREC_{cr}$. An implicit assumption is that the current source value remains inside the range. The answer fidelity of the answer $A : [L, U]$ for consumer request cr is defined as:

$$Fidelity(A, cr) = \begin{cases} 1 & L \leq V(cr.s, cr.t) \leq U \\ 0 & \text{otherwise} \end{cases}$$

where $V(cr.s, cr, t)$ is the current value of source s at t.

The effectiveness of handling consumer requests is measured using a QoS value, i.e., the probability of successful consumer requests. If consumer request does not state a preference, a successful consumer request not only finishes by the deadline (i.e., $TT_{cr} \leq RDL_{cr}$, where TT_{cr} is the total time taken to complete cr), but also delivers the answer at the desired accuracy; if consumer request indicates a bias towards timeliness/accuracy, we consider a consumer request to be successful if the deadline (accuracy) is met. Therefore, we define QoS as follows.

Definition 3 *The* **QoS** *of the system is defined as in Figure 2. 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.*

Since QoS is maximized when both deadline and accuracy constraints are met when no bias is specified, we set w_1 to be greater than w_2 or w_3; in addition, the ideal case is $QoS = 1$, i.e., all the requests satisfy their bias (if any). In our evaluation, we set $w_1 = 0.5$ and $w_2 = w_3 = 0.25$.

QoD is a metric of how "good" the data is. The metric varies from one system to another. In our system, the QoD is measured by data accuracy. We characterize data accuracy by *DS fidelity* and *DS validity*. The DS fidelity measures the divergence between stored range in the DS and the current source value. In contrast, the DS validity with respect to 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 an 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).

We define the DS fidelity of source s with current value V at time t and stored DS interval (L, U) as:

$$QoS = \begin{cases} w_1 \cdot \frac{\|\{cr_j | Bias_{cr_j} == 0, TT_{cr_j} \le RDL_{cr_j}, Fidelity(A_{cr_j}) == 1, PREC_{A_{cr_j}} \ge PREC_{cr_j}\}\|}{\|\{cr_j | Bias_{cr_j} == 0\}\|} & \text{(for req. without bias)} \\ + w_2 \cdot \frac{\|\{cr_j | Bias_{cr_j} == 1, TT_{cr_j} \le RDL_{cr_j}\}\|}{\|\{cr_j | Bias_{cr_j} == 1\}\|} & \text{(for req. favoring timeliness)} \\ + w_3 \cdot \frac{\|\{cr_j | Bias_{cr_j} == 2, Fidelity(A_{cr_j}) == 1, PREC_{A_j} \ge PREC_{cr_j}\}\|}{\|\{cr_j | Bias_{cr_j} == 2\}\|} & \text{(for req. favoring accuracy)} \end{cases}$$

Figure 2: **Characterizing system QoS in terms of timeliness/accuracy satisfaction of incoming requests**

$$FI_{ds}(s,t) = \begin{cases} 1 & \text{if } L \le V \le U \\ 0 & \text{otherwise} \end{cases}$$

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 percentage of time during T that s is faithful. If we assume that during T, s is uniformly accessed, then the probability of accessing a fresh value of s is equal to the percentage of time that s is faithful [10], that is $p_{fi}(s,T) = FI_{ds}(s,T)$. Hence, we can define the aggregate DS fidelity over all sources during the entire time period T as below:

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

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 if the DS precision of source s meets the cr_i's precision expectation at t. If the stored DS interval is (L,U), then the DS validity is defined as follows:

$$VA_{ds}(cr_i(s,t)) = \begin{cases} 1 & \text{if } PREC(L,U) \ge PREC_{cr_i} \\ 0 & \text{otherwise} \end{cases}$$

If there are $k > 0$ consumer requests accessing s at time t, then the DS validity for source s at time t

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

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:

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

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 overall* **QoD** *or Aggregate DS Accuracy $p_{aac}(S)$ is defined as follows:*

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

3.2. Problem Statement

Formally stated, given a set of n sources $S = \{s_1, ..., s_l\}$ and an input instance (request set) I, which is a collection of m incoming source update requests and n consumer requests $I = SR \cup CR = \{sr_1, ..., sr_m; cr_1, ..., cr_n\}$, our objective is to maximize EoS (Efficiency of the System)

$$\boxed{EoS = \frac{QoS \cdot QoD}{Cost}}$$

Where source update request, consumer request, QoS and QoD are respectively defined in Definitions 1, 2, 3 and 4. The communication overhead involved in the process of serving all the requests is represented by $Cost$ which is the average number of messages exchanged per request. The cost includes all messages exchanged for maintaining the DS and for additional probing that may service the accuracy requirements of consumer requests . Therefore, our objectives are to

• increase overall QoS indicated by the probability of successful consumer requests that meet deadline and/or accuracy requirements.

• increase overall QoD indicated by the probability of accessing accurate data in the DS.

• decrease overall Cost involved in the background process of maintaining the DS and also in serving requests.

In practice, due to highly dynamic system and network conditions, unpredictable application workloads, and frequently changing information sources, the joint optimization of these three factors is very complicated. Therefore, we aim to find good heuristics that addresses the tradeoff between timeliness (QoS), accuracy (QoD) and overhead. Our approach attempts to achieve end-to-end QoS by framing the tradeoff as two sub-problems. We manipulate QoS by proposing an algorithm to schedule incoming source update requests and consumer requests, aiming to maximize their accuracy and deadline constraints, hence desired QoS is achieved. This is done under the assumption that the directory service is maintained reasonably accurate. Simultaneously, we focus on adjusting QoD by presenting an efficient directory service maintenance algorithm that works in concert with the scheduling mechanism. The DS maintenance algorithm focuses on maximizing QoD without increasing the management overhead. Combining these two algorithms, the overall EoS is expected to improve and our experiments show that the proposed approaches result in very good EoS. In the following section, we present in detail techniques for request scheduling and DS maintenance.

4. Timeliness-Accuracy Balanced Scheduling

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 (QoS) against the need to keep the DS entry that reflects the source up-to-date (QoD). Missing deadlines might mean missing opportunities, operating on stale data might mean making wrong decisions. The scheduler must therefore determine the order in which the incoming source and consumer requests are handled so as to maximize 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) [1, 6]. There are some basic issues in mapping RTDB solutions to the information collection scenario. Firstly, in the RTDB context, consumer requests correspond to transactions (a series of read/write requests), which typically take longer time to be processed. Secondly, source update requests in the RTDB context do not have deadlines associated with them, enabling them to be treated differently from real-time transactions. 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 scheduling ordering to both types of requests is needed. 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. Specifically, TABS addresses the following issues to obtain a balance between QoS and QoD: (1) Decide on an ordering of the incoming source update requests (2) Decide on a relative ordering of source update and consumer requests. Based on their periodicity and deadline, we classify all incoming source update and consumer requests into four categories: periodic deadline based (P-DL), periodic non-deadline based (P-NODL), aperiodic deadline based (AP-DL) and aperiodic non-deadline based (AP-NODL). There are many different applications that can be classified into these request types. Even for the same type of application, the request types may vary due to environmental changes. For 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 non-deadline based* query is issued to the system; interested users may issue *aperiodic non-deadline based* queries to check the density irregularly. When the density of the toxic chemical is above certain threshold, *aperiodic deadline-based* 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* queries may be issued to provide timely and accurate density level information to aid emergency response teams in mitigating the hazard.

The objective of scheduling source update requests is to determine dynamically an update schedule which maximizes the overall QoD. 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). The remaining requests will be processed with lower priority (for archival purposes). We then prioritize update requests from different sources based on the popularity of the source, urgency and deadline of the request.

The problem of scheduling a mixed set of hard periodic tasks and soft aperiodic tasks in a dynamic environment has been widely considered when periodic tasks are executed under an EDF algorithm [4, 5, 7]. The goal of such EDF based joint scheduling algorithms has been (a) to meet all the deadlines of periodic tasks and (b) to minimize the average response time for aperiodic tasks. A thorough comparison of several scheduling techniques in terms of performance, schedulability and implementation complexity is presented in [16]. Among those studied techniques, the Total Bandwidth Server algorithm (TBS) exhibits superior overall performance and low implementation complexity. It assumes that all periodic tasks have hard deadlines (i.e. their periods), and all aperiodic tasks do not have deadlines.

TABS considers real-time joint scheduling of source update requests and consumer requests using the TBS algorithm as a basis and assigns an absolute deadline for each incoming request. The following assumptions and terminology are used in TABS:

• each periodic request r_i has a constant period PER_i and a constant worst case execution time E_i;

• all periodic requests $r_i : i = 1, ..., n_p$ have deadlines either as specified RDL for P-DL or as its period PER for P-NODL;

• the arrival time of the l^{th} periodic instance is given by $AR_i(l) = AR_i(l-1) + PER_i$;

• the absolute deadline of the l^{th} 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 $ard_j : j =$

$1, ..., n_{apd}$ have relative deadlines RDL, so its original absolute deadline $OADL_{ard_j} = AR_{ard_j} + RDL_{ard_j}$;

• all aperiodic non-deadline based requests $arnd_k : k = 1, ..., n_{apnd}$ do not have 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 utilization factor for periodic requests as $U_P = \sum_{i=1}^{n_p} \frac{E_i}{\min\{RDL_i, PER_i\}}$, and the capacity of the total bandwidth server as U_{AP}. To improve the response time of aperiodic requests, we assign a possible earlier deadline to each aperiodic request by applying the TBS algorithm [16]. Therefore, each time an aperiodic request enters the system, we optimistically assign it a deadline assuming that the total bandwidth of the server (U_{AP}) can be allocated to that request immediately. When the k^{th} 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 then inserted into the ready queue of the system and scheduled by EDF [11], 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 server utilization U_{AP}, that is, the processor utilization of the aperiodic requests is at most U_{AP} [17]. Based on the discussion above, for each incoming request of the system, we set an absolute deadline by which the request must be completed. 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. The assigned priority value reflects the popularity of the requested source (POP_s) and urgency of the request (UR_{r_i}) and is calculated as: $PR_{r_i(s,t)} = W_{pop} * POP_s + W_{ur} * UR_{r_i}$. 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 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 servicer. Figure 3 shows outlines of the scheduling algorithm.

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 tradeoff. 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. When the request servicer accepts a *consumer request*, we must determine whether it is necessary to probe the source when current value in the DS is not accurate enough. We aim to support both timeliness and accuracy, and at the same time, tailor the results to the consumers' preference (defined as $Bias$) if both constraints cannot be satisfied at the same time. For details of how the concept of 'bias' is incorporated in the request servicer sub-component, interested readers may refer to [9].

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 [16]). We also know that

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

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

Directory Service Maintenance for TABS The directory service maintenance module is responsible for keeping the directory service (DS) accurate enough so that most of the consumer requests can be served directly by consulting the directory service without probing the sources; hence, both communication overhead and delay are reduced. Prior work has presented the following approaches: *Instantaneous Snapshot Based Information Collection(SS) [18]* that stores the status of the desired parameters (e.g. residue capacity of network nodes and server nodes) using an absolute value obtained from a periodic snapshot; *Static Interval Based Information Collection (SI) [2]* that partitions the capacity of the collected information into a fixed number of equal size intervals and represents the obtained values by corresponding indices of the intervals inside which the values fall; and *Dynamic Range Based Information Collection* that holds the monitored parameter in the information

```
TABS()
/* thread 1: to manage both queues */
for each incoming request r
{   $PR_r = W_{pop} * POP_s + W_{ur} * UR_r$;
    switch (request category)
    {
        case P-DL: /* Periodic with Deadline*/
            $ADL_r = t + RDL_r$; break;
        case P-NODL:/*Periodic without deadline*/
            $ADL_r = t + PER_r$; break;
        case AP-DL: /* Aperiodic with Deadline */
        case AP-NODL: /*Aperiodic without deadline*/
            $ADL_r = max(t, ADL_{ar_{j-1}}) + \frac{E_i}{U_{AP}}$;
            break;
    }
    call insert($r$);
}

insert($r$)
{   switch (request type)
    {   case (source update request $sr$):
            /* insert into source update request queue */
            /* so as to preserve EDF order */
            $SR = SR \cup \{sr\}$ such that
                $ADL_{sr_i} < ADL_{sr_{i+1}}$ or $PR_{sr_i} \geq PR_{sr_{i+1}}$,
                  $i = 1, 2, .., m-1$;
            break;
        case (consumer request $cr$):
            /* insert into consumer request queue */
            /* so as to preserve EDF order */
            $CR = CR \cup \{cr\}$ such that
                $ADL_{cr_j} < ADL_{cr_{j+1}}$ or $PR_{cr_j} \geq PR_{cr_{j+1}}$,
                  $j = 1, 2, ..., n-1$;
            break;
    }
}

/* thread 2: to dispatch requests in ready queues */
while (true)
{ /* select the request with the earliest deadline or the highest priority*/
    $r^* = \{r | ADL_r \leq ADL_{sr_i}, 1 \leq i \leq m$ and $ADL_r \leq ADL_{cr_j}, 1 \leq j \leq n\}$;
    call RequestServicing($r^*$);
}
```

Figure 3: **The TABS algorithm**

repository using a range with an upper bound U and a lower bound L; the range may be modified dynamically based on the sampled information. Among several previously proposed dynamic range based strategies , a throttle-based approach (TR) where ranges are increased or decreased exponentially using a pre-determined throttle factor was shown to perform well [8].

Traditionally, the goal of the DS maintenance algorithm is to find an efficient way to adjust collection parameters (sampling frequency and range size) so that desired information accuracy is maintained while minimizing the communication overhead. In our scenario, request timeliness adds another dimension of complexity into the problem. Our objective is to develop an algorithm that can cater to a variety of sources (e.g., sensors with very limited resources and intelligence, or more powerful network intermediate components like routers). A DS that closely reflects the change of real-world can lower the probability of probing sources for current exact values, thereby saving request processing time and increasing the chance of meeting the request deadlines. We propose a Minimized Cost DS maintenance algorithm(MC), a dynamic range based cost-driven approach with restricted sampling that balances the cost/accuracy/timeliness tradeoffs. The MC algorithm analyzes all the cost factors in the whole process of information collection and identifies an optimal condition to make sure the cost is minimized. In addition, a curve fitting approach is applied to decide whether the current interval should be relaxed, tightened or moved to a new position so that the changes in source values can be more accurately captured. This is driven by the concept of ‘fidelity’. Furthermore, the sampling frequency is reduced when the source value is stable for a long enough period or when the interval is big enough to contain most of the changes. Since the DS maintenance algorithm is not the focus of this paper, interested readers may refer to [9] for details.

Note that the proposed approach is cognizant of both the desired QoS and QoD. The scheduler, where TABS is implemented, aims to provide better QoS; the DS maintainer, where MC (a Minimized Cost DS maintenance algorithm) is realized, is responsible for data freshness (QoD); the request servicer sub-component 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, QoD is maintained by the DS maintainer to assist the scheduler in achieving better QoS. This is because QoD has a direct impact on the frequency of future source update requests and consumer requests, which affects the system load and schedulability, thus indirectly has an impact on QoS.

5. Performance Evaluation

We evaluate the performance of the proposed scheduling and DS maintenance techniques under a variety of system conditions and heterogeneous request patterns. We compare TABS with the following scheduling algorithms (proposed in the context of real-time databases [1]): (1) *Source update request First (SF)* applies the source update request when it arrives into the system; (2) *Consumer request First (CF)* applies source request updates only when no consumer requests are waiting; (3) *Split Updates (SU)* 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; and (4) *On-Demand source request updates (OD)* extends the CF policy by giving precedence to source update requests only if there exists a consumer request that encouters a stale object (i.e. an object for which there exists a source update that has not yet been applied).

We compare our proposed DS maintenance approach (the MC algorithm) with three other approaches mentioned in previous section that vary in sampling policy and data representation. They are instantaneous values (SS), static intervals (SI) and dynamic intervals (TR).

The Simulation Environment: We built a simulator that consists of the various components described in Figure 1, including the mediator (with the scheduling and DS maintenance sub-components), information sources and the DS.

The sources are characterized as follows. The number of sources varies from 25 to 200. Each source holds one exact numeric value, and the DS holds all the interval approximations (where appropriate). Source values are picked randomly and uniformly from the range $[-150, 150]$; source values are changed periodically (initially set to be every 100 milli-seconds): 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 value reach certain thresholds. With deadline-based requests, the 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 $(100ms, 50s)$. Aperiodic consumer request arrival is modeled as a Poisson process with arrival rate $10/sec$ and with inter-arrival 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. Accuracy constraints of consumer requests are modeled as deviations from a mean value [12].

Experimental Results: Our performance study evaluates the various policy combinations in terms of the overall EoS, QoS (Definition 3), QoD(Definition 4) and Cost (measured as the number of messages exchanged in the system). We study system robustness to source capabilities and application requirements, as well as system scalability. For more detailed performance results, interested readers may refer to [9].

Basic Performance Evaluation: Figure 4 shows the system efficiency under all possible policy combinations. We observe that the four combinations of policies (TABS+MC, TABS+SI, FCFS+SI, FCFS+MC) result in better EoS than the others. Among these, we observe that the (TABS+MC) combination consistently performs better. This is because TABS keeps a good balance between source update requests and consumer requests, thus rendering reasonably good QoS and QoD; MC keeps the directory service reasonably accurate while minimizing the maintenance cost. We observe in a system with frequent incoming requests from both sources and consumers, the simplistic FCFS exhibits good performance, since deciding an order in which requests should be dispatched and processed could introduce unwanted overhead. We also notice that the static interval (SI) policies perform better than SS or TR. This is because an interval based representation (as in SI) requires no updates when values fall within the interval; furthermore, since the interval is static, no adjustment of upper and lower bounds is needed which further reduces the overhead. Figure 5A indicates that the (TABS+MC) combination provides higher QoD. We also observe that with this combination (See Figure 5B), the DS effectively maintains the needed accuracy, reducing the number of source probes for consumer requests.

System Robustness Evaluation: In our basic performance evaluation, we assume all the sources are unintelligent or *passive* and that they only respond to probes, i.e., when the mediator sends queries for their current values, the sources return the values, however they do not generate value reports spontaneously. In the real world, sources may possess the intelligence to report changes independently. This additional intelligence relieves the mediator from the burden of periodic sampling, however, it can also render very poor system performance when the frequency of source reports does not accurately reflect the changes in the source value.

Figure 6 (left) illustrates how the four best policy combinations adapt to source heterogeneity in the following three

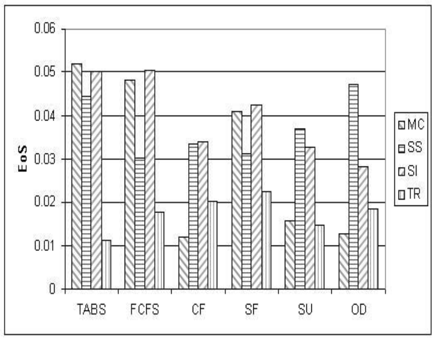
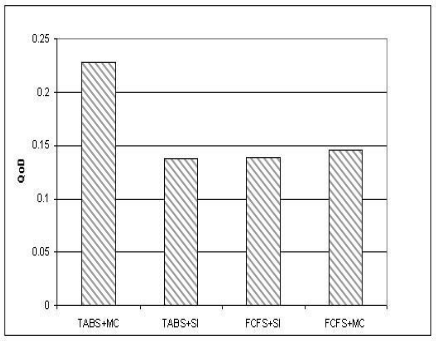
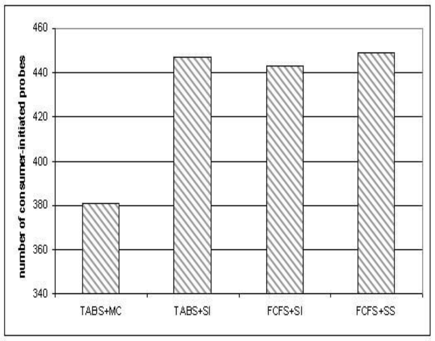


Figure 4: **EoS comparison**



A. QoD comparison



B. The number of consumer-initiated probes

Figure 5: **Detailed comparison of four best policy combinations**

categories: (a) all of the sources are passive (all-passive-sources); (b) half of the sources are smart and the other half are passive (mixed sources); (c) all sources are smart. As the number of intelligent sources increases, the performance of (TABS+MC) improves. When all of the sources are intelligent, it clearly outperforms the other combinations.

Figure 6 (middle) depicts system performance with different categories of consumer request deadlines. In the first category, *dl-small*, all the deadlines are uniformly distributed in the range of 500 ms to 1 second; with *dl-mixed*, all the deadlines are uniformly distributed from 500 ms to 1 minute; with *dl-big*, all the deadlines are uniformly distributed from 50 second to 1 minute. We observe that when all the deadlines are very small, none of the combinations exhibit very high EoS, since DS accesses themselves cause deadline violations. As the deadlines get bigger, EoS increases greatly, (TABS+MC) exhibits the highest EoS, since it schedules the consumer requests and source update requests in a balanced way. It probes the sources only when necessary (i.e., when the DS does not have accurate data and the deadline will not be violated) and also maintains the DS accurately while minimizing the cost involved.

System Scalability Evaluation: Figure 6 (right) shows the performance of the policies with increasing number of sources. In addition, we show the benefits of gradually adding intelligence to scheduler and DS maintenance module. The EoS is significantly higher as more intelligence is added to each component, as illustrated by the superior performance of the (TABS+MC) policy. (The (FCFS+SS) represents the simplest policy combination. Adding more intelligence to scheduler (i.e., replacing FCFS with TABS) improves EoS since TABS ensures fairness among the requests and effectiveness of handling each request. In addition, adding intelligence to DS maintainer (i.e., replacing SS with MC) decreases the overhead involved in maintaining the DS, thus further increasing the EoS.) We observe that as the number of sources in the system increases, the EoS decreases slowly since the system is busy handling more source update requests. In other experiments, we also observe that the (TABS+MC) policy exhibits superior performance with increase in the number of consumer requests.

In summary, our performance studies indicate that the policy combination of (TABS+MC) exhibits highest system efficiency, this implies that adding intelligence to the mediator modules is beneficial. 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, this combination is very robust to source heterogeneity and variations in consumer request deadlines.

6. Related Work and Conclusions

In this section, we review existing work in the areas of event-driven/real-time middleware and data management and compare it with our work.

The CORBA event service introduces the concept of event channel, supplier and consumer. In addition, there exist other CORBA-based event services such as TAO [13] and COBEA(CORBA-based Event Architecture) [14], as well as non CORBA-based publisher/subscriber service architecture such as SIENA [3] and CMU Pub/Sub [15]. The architecture we proposed in this paper is different from the existing work reviewed above in several ways. Firstly, unlike the event channels in publisher/subscriber architectures, our mediator component supports specification of QoS constraints (coarse and fine-grained timeliness requirements, and accuracy constraints) from both publishers (sources) and subscribers (consumers). The mediator also supports both periodic and aperiodic requests. Furthermore, the mediator component incorporates a sophisticated repository management sub-component, which maintains the repository at a certain accuracy level so that desired timeliness and accuracy constraints from consumers can be achieved. Languages and methodologies for event composition, which is inherent in event-based middleware, compliments our effort and can be incorporated as an additional layer/service in the mediator component.

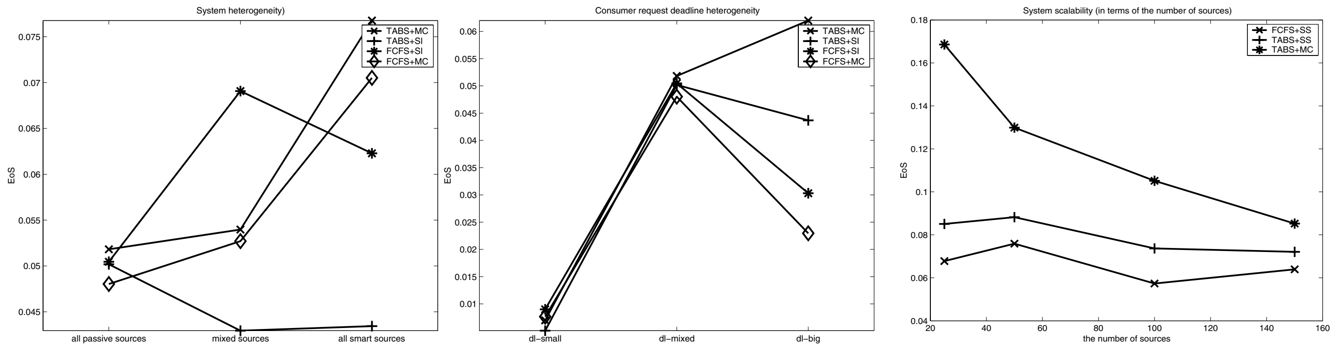


Figure 6: **System heterogeneity and scalability**

Several projects address the tradeoffs between *transaction timeliness and data freshness* such as the Stanford Real-Time Information Processor (STRIP) project [1], [6] and the QMF project [19]; The tradeoff between *data accuracy and cache maintenance overhead* is addressed by approximate data caching [12]. Our work addresses the tradeoffs in balancing all three factors- *timeliness, accuracy and overhead* in information collection for dynamic distributed real-time environments. We also propose a middleware framework within which the developed algorithms can be implemented. Our real-time joint scheduling of consumer requests and information source update requests, combined with our cost-effective DS maintenance algorithm perform very well under different system load and input characteristics. The design details of a full-fledged prototype system that implements the proposed architecture and the implementation issues therein are beyond the scope of this paper (See [9] for more details). We believe that adaptive middleware techniques for service management such as those described in this paper are key to guaranteeing application QoS in distributed real-time environments.

References

[1] B. Adelberg, H. Garcia-Molina, and B. Kao. Applying update streams in a soft real-time database system. In *ACM SIGMOD*, 1995.

[2] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi. Quality of service based routing: A performance perspective. In *ACM SIGCOMM*, 1998.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3), Aug. 2001.

[4] H. Chetto and M. Chetto. Dynamic scheduling of real-time tasks under precedence constraints. *IEEE Trans. on Software Engineering*, 15(10), 1989.

[5] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *The Journal of Real-time Systems*, 1990.

[6] A. Datta and I. Viguier. Providing real-time response, state recency and temporal consistency in databases for rapidly changing environments. *Information Systems*, 22(4), 1997.

[7] T. Ghazalie and T. Baker. Aperiodic servers in a deadline scheduling environment. *Jl. of Real-time Systems*, 1995.

[8] Q. Han and N. Venkatasubramanian. Autosec: An integrated middleware framework for dynamic service brokering. *IEEE Distributed Systems Online*, 2(7), 2001.

[9] Q. Han and N. Venkatasubramanian. Addressing timeliness/accuracy/cost tradeoffs in information collection for dynamic environments. Technical report, University of California-Irvine, 2003. http://www.ics.uci.edu/~qhan/papers/rtic-techrep.pdf.

[10] A. Labrinidis and N. Roussopoulos. Update propagation strategies for improving the quality of data on the web. In *the 27th VLDB Conference*, 2001.

[11] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, 20(1), 1973.

[12] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *ACM SIGMOD*, 2001.

[13] C. O'Ryan, D. C. Schmidt, and J. R. Noseworthy. Patterns and Performance of a CORBA Event Service for Large-scale Distributed Interactive Simulations. *Intl. Jl. of Computer Systems Science and Engineering*, 17(2), Mar. 2002.

[14] P. Pietzuch, B. Shand, and J. Bacon. A Framework for Event Composition in Distributed Systems. In *the Middleware 2003 Conference*, 2003.

[15] R. Rajkumar, M. Gagliardi, and L. Sha. The Real-Time Publisher/Subscriber Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation. In *IEEE RTAS*, 1995.

[16] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-time Systems*, 10(2), 1996.

[17] M. Spuri, G. Buttazzo, and F. Sensini. Robust aperiodic scheduling under dynamic priority systems. In *RTSS*, 1995.

[18] M. Stemm, R. Katz, and S. Seshan. A network measurement architecture for adaptive applications. In *InfoCom*, 2000.

[19] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. Controlware: a middleware architecture for feedback control of software performance. In *ICDCS*, 2002.

IEEE COMPUTER SOCIETY