# Addressing Scheduling Issues in Context Information Management

M. Chantzara[*], N. Vardalachos[**] and M. Anagnostou[*]

[*]School of Electrical & Computer Engineering, NTUA,
[*]9 Heroon Polytechniou Str, 157 73 Zografou, Athens, Greece, {mchantz, miltos}@mail.ntua.gr
[**]Department of Electronic and Electrical Engineering, UCL,
[**]Torrington Place, London, WC1E 7JE, UK, nvardalachos@iee.org

## ABSTRACT

The provision of context-aware services, that is crucial for mobile and ubiquitous computing, prerequisites appropriate management methods for acquiring, processing and distributing context information. This paper focuses on the challenge that refers to the seamless provision of real-time access to the dynamically changing context data. It studies the problem of handling the order of the information update requests issued by the context sources. The objective is to maximize the freshness of the delivered data and satisfy the timing constraints of the context consumers, when firm deadlines, non-pre-emptive order and limited processing resources are considered. In order to tackle the conflicting requirements, the Importance/ Popularity/ Urgency - Aware Scheduling Algorithm is developed. The proposed algorithm is evaluated through extensive performance studies.

***Keywords***: Context information, context management, scheduling algorithm, timing constraints, data freshness.

## 1 INTRODUCTION

The advances in wireless communications and user mobility have given quite a boost to the research about new classes of systems called pervasive systems [1]. The creation and provision of the Context-Aware Services (CASs) that get aware of the execution environment such as location, time, user's activities, devices' capabilities in order to tune their intended functionalities and adapt to both the changing environment and the user requirements is of great importance [1] towards the materialization of pervasive computing. One of the key challenges of the field is "*the seamless provision of real-time access to the dynamically changing context data*". "*Seamless*" points to the design of middleware solutions for distributing of context data. Distinguishing the functionality of CASs from the acquisition of context provides flexibility and scalability. As stated in the [2], the role of the context management middleware is similar to that of the database management, only one level higher. The context middleware is responsible for allowing one or more users/services to create and/or access data stored in distributed databases or dynamically produced by complicated tools and sensors. This challenge also talks about "real-time access". However, "real-time" should not be confused with "fast", since the objective of real-time computing is to meet the indicated timing requirements of each task [3][4]. It is a fact that missing deadlines might mean missing opportunities as well as operating on stale data might mean wrong decisions for the context-aware services. Therefore, data freshness and requests' timeliness should be balanced.

In order to tackle the aforementioned issues, this paper proposes a system of brokers that is responsible for acquiring the information from the sources, efficiently processing requests for information and distributing the data to the requestors. It then focuses on the problem of scheduling the sources' data updates and the requests for information, considering both the timing and the resource constraints. Requests' timing constraints include deadlines, and requests' must be scheduled such that these constraints are met. Data freshness is the data temporal consistency, describing how old a data item can be and still be considered valid. Solutions for addressing the pertinent scheduling issues have been developed in the context of real-time databases ([5][6]), that are asked to maintain time-constrained data and time-constrained transactions. This paper introduces the scheduling algorithm, called ***I**mportance/ **P**opularity/ **U**rgency- Aware Scheduling Algorithm (IPU)* for determining the order in which the incoming source update requests are handled, considering that the requests have firm deadlines. The objective is to minimize the missed deadlines and maximize the freshness of the delivered data. The proposed algorithm is evaluated through extensive performance studies.

The rest of the paper is organised as follows: Section 2 describes the context information collection model. Based on this model, in Section 3 the problem of establishing priorities among the update requests is formulated and the proposed algorithm is presented. The evaluation setup, the metrics that are used and the performed simulations are presented in Section 4. The related work is discussed in Section 5. Finally, Section 6 concludes the paper.

## 2 CONTEXT DISSEMINATION MODEL

Aiming to make the creation and provision of CASs easier and more efficient, the abstraction between context information sources and CASs has been widely proposed in the literature [7]. Based on this concept, the role of Context Broker (CB) is introduced. The CB is responsible for handling the collection of the information from the sources, efficiently processing requests for information and distributing the context data to the requestors. The IST project CONTEXT [8] followed this model and

implemented a context management middleware that is based on Peer-to-Peer architecture and supports the cooperation of multiple peer CBs. The distributed CBs act as a federation that cooperates to answer context requests. A detailed description of the communication mechanisms between the CBs is presented in [9]. Each CB offers the *Context Information Provider Interface* (CI Provider API) that enables context sources to initially declare and then supply the information they produce and the *Context Information User Interface* (CI User API) that enables CASs to request the information they want to consume. The same interfaces can also be used by the sources that act as logical sensors, process data produced by other sources, and provide high-level data. For the rest of the paper, the term context consumer describes every entity asking for context information.

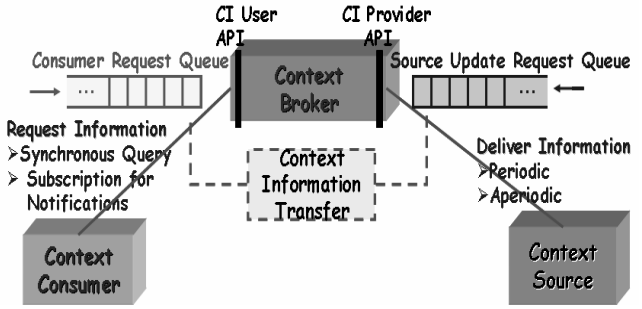


Figure 1: Context Information Collection Model

The CB receives two types of requests: *Source Update Requests* issued by the context sources and *Consumer Requests* issued by the consumers. All the requests are maintained to the corresponding request queue (*Source Update Requests Queue* and *Consumer Requests Queue*) until they are served by the Broker (Figure 1). Once a consumer request is dispatched, the proper source to provide the requested information is discovered based on the approach described in [10]. The consumers only ask for current data, not historical data at a specified time instant. The CB supports both synchronous and asynchronous means of communication in order to satisfy both synchronous queries and asynchronous event notifications. Therefore, the context consumers could either issue a context query or a subscription for context events. In the latter case, the consumer is subsequently notified of any event generated by the pertinent source. When it wishes to stop receiving such notifications, the context consumer issues a un-subscription.

Based on their periodicity, the context sources can be characterized as: 1) *Periodic Sources*: When they provide updates periodically, thus every specific time period that is called refresh time. 2) *Aperiodic Sources*: When they provide updates aperiodically (the sensed value changes or it reaches a given threshold). Regarding the context provisioning scheme, two types of information delivery, are considered: 1) *Passive Sources*: This type provides information in consumer-initiated pattern. 2) *Active Sources*: This type provides information in a source-initiated pattern. The context sources are also distinguished based on the offered "value" to the context consumer. This "value" is usually expressed by the purchase cost; Context data with higher cost are more "valuable" to consumers and producers as well as the CB that distributes it. Although data could have many different values, only three sets: low, medium and high importance, are considered.

When the CB receives a context request, it discovers the appropriate source to answer it. If the pertinent source is a passive one, it transfers the query to source. Once the source receives the request, it either delivers the current sensed value back to the consumer (if the request is context query) or initiates the delivery of the updated values in the form of notifications (if the request is context subscription). For the active sources, the CB is responsible for collecting the updated values and storing the delivered values in the context repository. Therefore, when the CB receives a context query for the data of an active source, it retrieves the latest stored value that is already delivered by the pertinent source and distributes it to the consumer. When the CB receives a subscription for notifications produced by an active source, it stores the subscription and distributes the context notifications that are generated by the pertinent source as soon as they arrive. Finally, a caching utility that stores the most recent collected values, to be later used, is developed. Answering requests with the cached values that still remain valid is expected to satisfy better the timing restrictions, since data will be provided in less time.

# 3 SCHEDULING PROBLEM

## 3.1 Problem Formulation

The collection and distribution of context information involves many complex problems such as data integrity, discovery, real-time update, secure storage, distribution, caching and replication. This paper is focusing on the problem of prioritising the issued requests in order to maximize the freshness of the delivered data and satisfy the timing constraints of the context consumers, while firm deadlines, non-pre-emptive order and limited processing resources of the CB are considered. Before formulating the problem, the parameters that characterise each type of requests are defined. Every request is described by:

1. *Start Time (S):* The time point it is issued.
2. *Arrival Time (A):* The time point it enters the queue.
3. *Execution Time (E):* The time point it is executed.
4. The *Deadline* is the latest time point that the request should be served, otherwise it is considered useless. This parameter is defined in two ways:
   - *Relative Deadline (RD):* It is the time period after the *S*.
   - *Absolute Deadline (AD):* It is the time point that comes out of adding *RD* to *S*.

A Consumer Request *(CR)* is also characterised by:

1. *Selected Context Source (SS(CR)):* The context source to retrieve the requested info.
2. *Source Update Request SUR(CR)*: It delivers the value that is utilized to answer the context request.

A Source Update Request *(SUR)* is also characterised by:

1. The *Context Source CS(SUR)* that produces the update request. The quality characteristics of a *CS* are: Accuracy

*A(CS)*, Refresh Time *RT(CS)* and the Cost *C(CS)*, while the timestamp of the latest stored value is *T(CS)*.

2. *Importance I(SUR):* As it was mentioned previously, three importance levels: low, medium, high, taking values 1,2,3 respectively, are defined based on the cost of the context information.

Regarding the data freshness and the consumer request timing restrictions, the following definitions are introduced:

- *Definition 1:* Consider the *SUR* issued from the source *CS*. The delivered data is considered fresh, if it is consumed until *AD(SUR)=RR(CS(SUR))*.
- *Definition 2:* Consider the *CR* for data from the source *CS*, namely *SS(CR) = CS*. *SUR* is the most recent source update request. The following statements hold for the *CR*:
  - It receives fresh data, if it is served until *AD(SUR)*.
  - It meets its deadline, if it is served until *AD(CR)*.
  - It is successful, if it is served in time with fresh data. Thus, it is served until *min{AD(CR),AD(SUR)}*.

The scheduling problem that is studied consists of three sub-problems. Given the timing and resource constraints, it should be determined: 1) What source update request to install next. 2) What consumer request to answer next. 3) What is the priority of a source update request versus a consumer request. However, since the workload imposed by the consumer requests is minimal, comparing to the workload imposed by the source updates requests, the problem can be deduced to the first sub-problem which refers to the scheduling of the source updates requests. This happens because, in a context provision system, most of the consumer requests ask for context notifications during a specific time period, while context queries are less preferable. When a context subscription is issued, the CB is informed accordingly and delivers the proper context notifications upon arrival. When a context query is issued, the stored context value is retrieved and it is delivered to the requestor. On the other hand, the delivery of source update requests requires more system resources for both discovery and storage processes. Therefore, it is assumed that both context subscriptions and queries are dispatched immediately when entering the Consumer Request Queue, before dispatching any source update request. Finally, the objective and the restrictions of the resource-constrained and time-constrained scheduling problem are:

- ***Objective:*** The maximization of the number of the consumer requests that are successfully answered.
- ***Restriction 1:*** The number of requests that the broker can dispatch is limited by its processing resources (CPU).
- ***Restriction 2:*** Only the source update requests that deliver fresh data should be dispatched.
- ***Restriction 3:*** Context queries should be answered immediately.
- ***Restriction 4:*** Context notifications should be delivered within the deadline the context consumer that subscribed.

## 3.2 Proposed Scheduling algorithm

The scheduling algorithms can be characterised as being either static or dynamic. A static approach pre-determines the schedules for the system, while a dynamic method determines schedules at run-time. Some of the state-of-the-art algorithms are: 1) *First In First Out (FIFO)* that assigns priorities based on the time point they arrive to the system. 2) *Rate Monotonic (RM) that* assigns priorities inversely proportional to the period. 3*) Earliest Deadline First (EDF)* that assigns priorities based on the absolute deadlines.

This paper introduces the dynamic scheduling algorithm called *I*mportance/*P*opularity/*U*rgency-Aware Scheduling Algorithm (IPU) for scheduling the source update requests. This algorithm considers both the timeliness requirements of a request and how valuable is the request to the system [11]. Since the concern is not only how many transactions are missed, but also which transactions are missed, the proposed algorithm considers the transactions' worth expressed by the importance of the sources. Moreover, in order to minimize the requests that miss their timing constraints the "popularity" of a source update request, describing the demand of it, is considered. As a result of this, requests with higher demand are given higher priority than the others. The popularity *Pop(SUR, t)* of the source update request *SUR* expressed by the context source *CS* is computed by the access ratio of *CS* at given time *t*.

$$Pop(SUR,t) = \frac{\#CR_i, A(CR_i) \le t, SS(CR_i) = CS(SUR)}{\#CR_i, A(CR_i) \le t}$$

Furthermore, since the objective of the system is to deliver fresh context values, data timeliness is considered for both context notifications and queries. Therefore, the "urgency" is introduced in order to describe the importance of dispatching the source update requests, based on the timeliness requirements. It could be defined that urgency is inversely proportional to the request absolute deadline, just like EDF, but in that case, even though the context notifications would be delivered within their deadline, the stored values would soon expire and the context queries would be answered with stale data. Therefore, it is more valuable to dispatch the requests in such an order that the stored values remain fresh for a longer period. Additionally, the IPU is trying to do the ordering fairly enough among the context sources. Thus, the refresh cycles that no updated value has been stored are taken into account. The Urgency *Urg(SUR, t)* of the source update request *SUR*:

$$Urg(SUR,t) = UrgImp(SUR,t) * (S(SUR) + AD(SUR) - t)$$

The UrgImp(SUR,t) describes the refresh cycles of the corresponding context source that no source update request is dispatched. It ensures that any deadline misses are scattered across the different context sources. For the periodic source update requests, this parameter is defined:

$$Urg\ Imp(SUR,t) = \frac{S(SUR) - T(CS(SUR))}{AD(SUR)},$$

while for the aperiodic ones it is a system parameter. The value assignment to this parameter represents the preference factor between periodic and aperiodic sources.

To sum up, the IPU assigns priorities of the source update requests dynamically based on the Urgency, the Importance and the Popularity. The necessary data to compute the priority of each source update request is available by both

the Broker and the request itself. The IPU enables more source update requests to be served in time, so that more consumer requests are answered successfully. Finally, the Priority *PR(SUR,*t) of the *SUR* at *t* is computed based on the following equation that involves no extra processing than simple math operations:

$$\Pr(SUR,t) = Pop(SUR,t) * Urg(SUR,t) * I(SUR)$$

# 4 EVALUATION

This section presents the experimental setup, the performance metrics and the performed simulations. The proposed algorithm is evaluated against the algorithms: FIFO, EDF and RM.

## 4.1 Simulation Model

The system model is described in terms of the context sources, the source update and consumer requests, and the system parameters:

 ***Sources and Source Update Requests Model:*** Consider N number of context sources ($N_{sources}$). If the probability for a source to be active is $p_{active}$, the probability for a source to be passive ($p_{passive}$) would then be $1\text{-}p_{active}$. Similarly, the probability for a source to provide periodic and aperiodic updated context values would respectively be $p_{periodic}$ and $p_{aperiodic}= 1\text{-}\ p_{periodic}$. Also, the respective probabilities for a context source to provide low, medium and high value data are $p_{low}$, $p_{medium}$ and $p_{high}=1\text{-}p_{low}\text{-}p_{medium}$. The refresh cycle of the context sources is a uniformly distributed integer in the range *(0, 2* Refresh_Time)*, with mean value *Refresh_Time* (*RT*). All the sources are initialized before the consumer makes any requests. The periodic sources provide a new context notification every refresh cycle, while the aperiodic sources generate aperiodic context notifications after a specific number of refresh cycles. This number is an exponentially distributed integer with a mean *Aperiodic_Rate* (*AR*). Since the context sources are located at different network nodes, the source update requests do not arrive at the Source Update Request Queue immediately when produced, but with an arrival latency described by a percentage of the Refresh Rate (*Delay_Ratio*). Thus, for the source update request SUR the arrival time at the queue is: *A(SUR)=S(SUR)+Delay_Ratio*RT(SUR)*. Finally, the estimated mean execution rate for the source updates that describes the number of update requests that are served within one second is known as the *Execution_Rate* (*ER*).

 ***Consumer Requests Model:*** Consider N number of consumer subscriptions ($N_{consumers}$) for context notifications. Apart from these subscription requests, there are queries for the current context values. These arrive at the Consumer Request Queue following a Poisson distribution with a mean inter-arrival time of *Queries_Rate* (*QR*) seconds. The demand for the context data is produced randomly and uniformly from the context sources. The deadline of each context consumer is computed based on the *Deadline_Ratio* (*DR*), describing the percentage of the refresh rate for a given source. Thus, for a context consumer CR the Absolute Deadline is equal to *AD(CR) = A(CR) + Deadline_Ratio * RR(CS(CR))*.

 ***System Model:*** The time window (*Abortion_Window*) describes the time a given source update request is allowed to wait in the queue before being dispatched. The *Queue Size* describes the capacity of the Source Update Request Queue, i.e. the number of source update requests that are allowed to wait in the queue. Finally, the *Time* describes the time period the system is being monitored.

In order to quantify the workload of the system, the parameter *Load* is used. This parameter describes the ratio of the work generated to the total execution capacity. Obviously, the focus is on the source update requests, since the consumer requests produce no load. The arrival rate of the updates generated by a periodic source is *1/RT*, while the arrival rate of the updates generated by the aperiodic ones is *1/RT*AR*. Therefore, the Load is defined as follows:

$$Load(\%) = \frac{(\frac{1}{RT} * p_{periodic} + \frac{1}{RT * AR} * p_{periodic}) * Nsources}{ER} * 100$$

## 4.2 Performance Metrics

In order to evaluate the algorithm's performance, the following performance metrics are introduced:

♦ The ***Delivered Notifications Ratio*** is the average fraction of the updated context values delivered to the context subscribers.

♦ The ***Fresh Queries Ratio*** is the fraction of the queries that were answered with fresh data retrieved from the cache or the context repository.

♦ The ***Profit*** that is gained by the CB for delivering fresh updates in time and answering queries with fresh data.

## 4.3 Simulation Results

In order to evaluate the effectiveness of the proposed algorithm, several test scenarios have been developed. In each of them, the effect of one system parameter to the performance metrics is examined. Due to space limitations, only some of the performed tests are presented in this paper. Each experiment has been evaluated with 10 samples, and apart from the average value, the 90 percent confidence is also reported. The default system parameters are depicted in the Table 1. Based on these parameters, the Load is 100%.

Table 1: Default Simulation Parameters

| ***Parameter*** | ***Value*** | ***Parameter*** | ***Value*** |
|---|---|---|---|
| `Nsources` | 200 | `Execution_Rate` | 12 updates/sec |
| $P_{active}$ | 50% | `Refresh_Time` | 10 sec |
| $P_{passive}$ | 50% | `Aperiodic_Rate` | 5 refresh cycles |
| $P_{aperiodic}$ | 50% | `Nconsumers` | 400 |
| $P_{periodic}$ | 50% | `Queries_Rate` | 5 sec |
| $P_{low}$ | 33% | `Deadline_Ratio` | 1 (for notifications)<br>0 (for queries) |
| $P_{medium}$ | 33% | `Abortion_Window` | Refresh_Rate |
| $P_{high}$ | 33% | `Time` | 1000sec |
| `Delay_Ratio` | 0.1 | `Queue_Size` | ∞ |

***Test 1 - "Effect of the Refresh Rate":*** In this test case, the effect of the refresh rate is evaluated. Thus, for sources having mean Refresh Time RT={5, 10, 15, 20, 25, 30}, the three performance metrics (Delivered Notifications Ratio, Fresh Queries Ratio, Profit) are reported in Figure 2, Figure 3 and Figure 4. For the considered values of mean Refresh Time, the respective % Load of the system is {200, 100, 50, 25, 12.5, 6.25}.

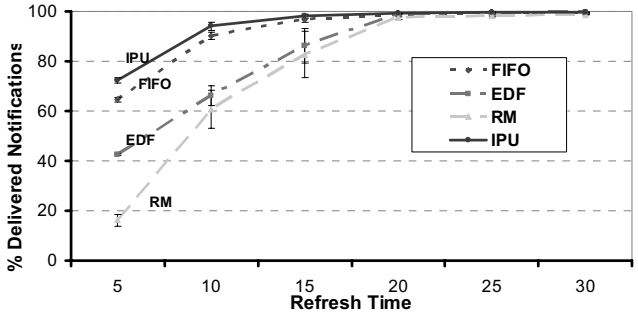


Figure 2: Delivered Notifications Ratio as a function of the mean Refresh Time

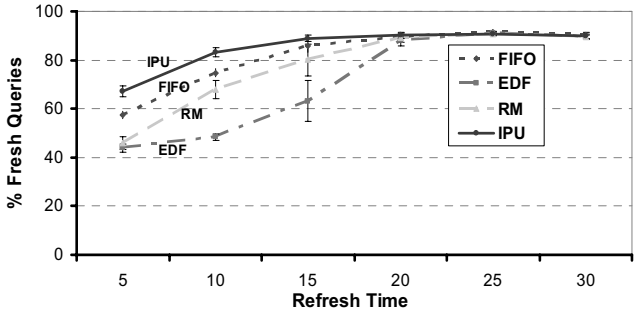


Figure 3: Fresh Queries Ratio as a function of the mean Refresh Time

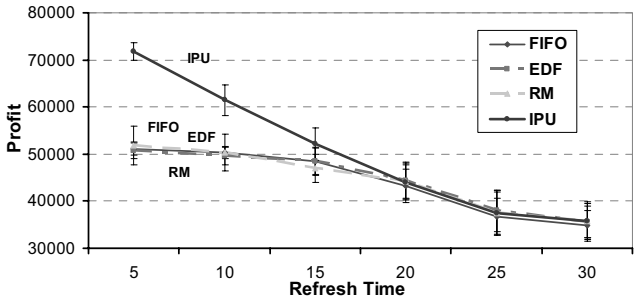


Figure 4: Profit as a function of the mean Refresh Time

As it can be seen from the previous figures (Figures 2-4), when the Refresh Time is low, the proposed algorithm performs better than the state-of-the art ones, while when it gets higher the performance metrics of the algorithms tend to converge. For mean Refresh Time higher than 20 sec (when the load is smaller than 25%) the four algorithms have similar performance. It should be noted that the EDF achieves the smaller Fresh Queries Ratio due to the fact that the source update requests are dispatched close to the expiry time of the delivered data. As it can be deducted from the Figure 4, the IPU achieves high profit even in overload conditions; for 200% Load, the IPU achieves about 50% higher profit than the other algorithms.

***Test 2 - "Effect of Execution Rate":*** In this test case, the effect of the Execution Rate is evaluated. Thus, when the Execution Rate becomes {3, 6, .., 27, 30} updates/sec, the algorithms' performance is shown in Figure 5, Figure 6 and Figure 7. For the considered Execution Rate, the % Load is {400, 200, 130, 100, 80, 67, 57, 50, 44, 40} respectively.

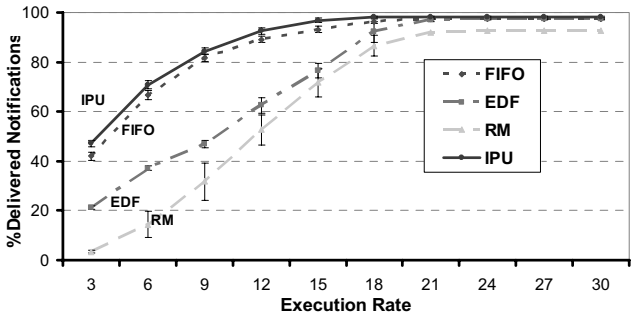


Figure 5: Delivered Notifications Ratio as a function of the Execution Rate

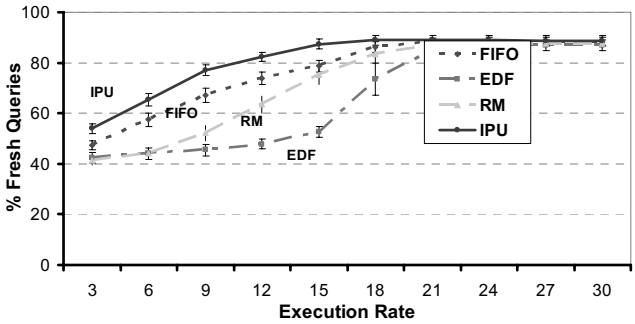


Figure 6: Fresh Queries Ratio as a function of the Execution Rate

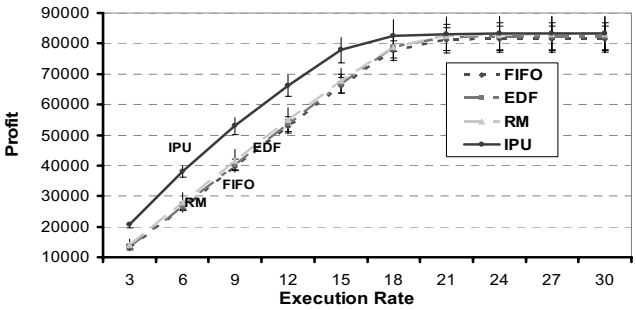


Figure 7: Profit as a function of the Execution Rate

As it can be seen from the previous figures (Figures 5-7), when Execution Rate is low, the IPU outperforms the state-of-the art ones, but when it increases, the performance metrics of the algorithms tend to converge. The remarks made for the Test 1 are also confirmed in this experiment.

***Test 3 - "Effect of the $p_{periodic}/p_{aperiodic}$ sources":*** In this test case, the effect of the number of the sources that provide aperiodic/ periodic updates is evaluated. For the {(20%, 80%), (50%, 50%), (80%, 20%)} of $p_{periodic}$, $p_{aperiodic}$ respectively, the Delivered Notifications Ratio and the Fresh Queries Ratio for both the aperiodic and periodic are compute, when using the algorithms FIFO and IPU.

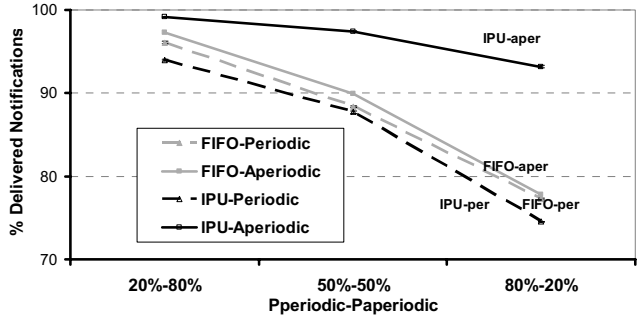


Figure 8: Delivered Notifications Ratio as a function of the $p_{periodic}$ - $p_{aperiodic}$

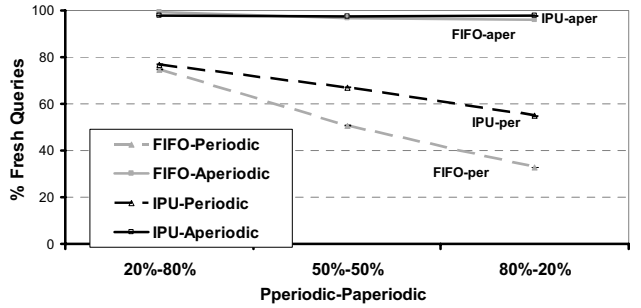


Figure 9: Fresh Queries Ratio as a function of the $p_{periodic}$ - $p_{aperiodic}$

As it can be seen from the Figure 8 and the Figure 9, as the number of sources providing updates periodically gets higher, the reported ratios get lower. This happens because the Load of the system gets higher and more source update requests are aborted. Nevertheless, the ratios referring to the aperiodic updates tend to decrease with slower rate (Delivered Notifications Ratio) or even remain stable (Fresh Queries Ratio) when utilizing the IPU for scheduling the source update requests. Furthermore, in every case the ratios of aperiodic are higher than the ones of the periodic. This happens even in the case that the percentage of the aperiodic sources is 80%, and is due to the facts that a request for aperiodic data is never answered with stale information and is considered successful even when there isn't any fresh notification to be delivered.

***Test 4 -"Effect of the $p_{low}/p_{medium}/p_{high}$ sources":*** In this test case, the effect of the number of the sources that provide low/medium/high updates is evaluated. For the {(40%, 40%, 20%), (33%, 33%, 33%), (20%, 20%, 60%)} of $p_{low}$, $p_{medium}$, $p_{high}$ respectively, the Delivered Notifications Ratio and the Fresh Queries Ratio for each type are computed. The results are depicted in the Figure 10 and the Figure 11.

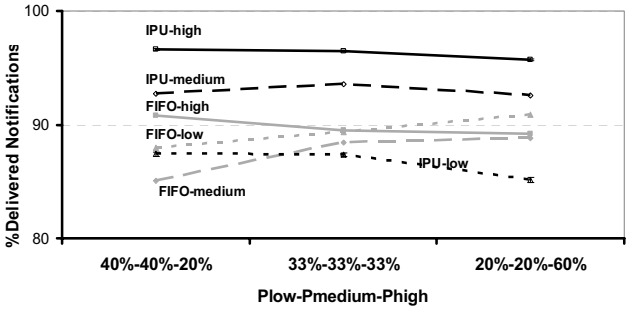


Figure 10: Delivered Notifications Ratio as a function of the $p_{low}$ - $p_{medium}$ - $p_{high}$

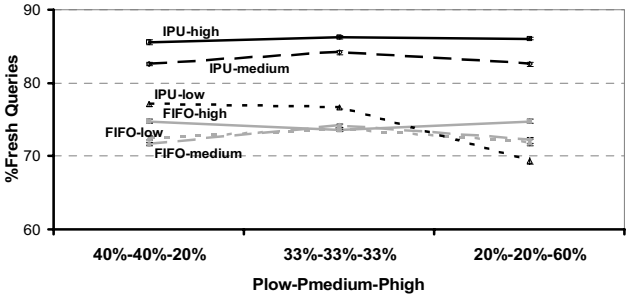


Figure 11: Fresh Queries Ratio as a function of the $p_{low}$ - $p_{medium}$ - $p_{high}$

As it is displayed in the previous figures (Figures 10-11), the IPU achieves higher ratios for the high valued requests and the medium ones, while the FIFO behaves almost the same for the three types. The difference of the IPU ratios is about 10% between the high-valued and the low-valued.

Finally, regarding the storage requirements of the algorithms, it should be stated that the average waiting time of the source update requests in the queue is minimized when the IPU is used while it is maximized when FIFO and EDF are used. Due to space limitations the corresponding diagrams are not displayed. On the other hand the computational complexity of IPU and EDF is higher than the complexity of RM and FIFO, since the first ones assign priorities dynamically.

## 5 RELATED WORK

A very good survey of the literature in the field of real-time data management is presented in [12]. The concerns that have been subject of research in this field are discussed and include: data, transaction and system characteristics, scheduling and transaction processing, distribution and quality of service and quality of data. The [13] concentrates on the theoretical research work in the field of real time scheduling in the past 25 years. It distinguishes the scheduling algorithm into three basic categories: Fixed-priority, Dynamic-priority and Feedback scheduling.

The authors of [14] were the first that discussed about the timeliness and freshness requirements in real-time databases. A soft real-time main memory database, called STRIP, having special facilities for importing and exporting data as well as handling derived data, is the product of their work. In order to balance the conflicting requirements, they presented four algorithms for scheduling sensor data updates and user transactions. The basic concern of these algorithms is the priority assignment between the user transactions and the updates transactions, while the baseline scheduling that is used is FIFO.

Data temporal consistency in addition to logical consistence is discussed in [16]. This paper studies two-phase locking and optimistic concurrency control algorithms and examines the performance of RM and EDF scheduling algorithms. The evaluation results showed that RM and EDF are close when load is low, but EDF outperforms at higher loads. The difficultly to maintain data and transaction time constraints motivated the authors of [17] to introduced the notions of "data-deadline" and "forced wait". Moreover, the concept of "data-similarity" is explored. These notions are used to enhance the baseline algorithms EDF and Least Slack First, and experimental evaluation showed that they improve their performance.

Applying the high level notions of the feedback control in managing database's performance is claimed to improve the database throughput and response time due to its robustness against unpredictable situations. According to the QMF architecture that uses it [18], the deadline miss ratio and new data freshness metrics are defined by the database administrator as the desired quality of real-time services for a specific application. In order to support the desired QoS and prevent overload, the QMF applies feedback control, admission control and flexible freshness management schemes. The [19] presents how this approach is applied to real-time e-commerce data services. User requests are classified into several service classes according to their importance, and they receive differentiated real-time performance guarantees in terms of deadline miss ratio. In the [20] another approach that uses both feedback control and imprecision control techniques is presented. According to it the update frequency of the data is calculated based on the imprecision of data and requests.

The [21] refers to real-time information collection, while the [22] addresses the scheduling issues that arise in this field. It talks about an information mediator that coordinates and facilitates communication between information sources. The objective is to maximize the efficiency of the system, which is defined by the probability of successful consumer requests, how good is the data and the communication overhead involved in the process of serving all requests. In order to tackle the arising tradeoffs, it uses the TABS scheduling algorithm that is based on the EDF and balances timeliness/accuracy and the MC directory service maintenance algorithm that tries to minimize the cost.

Context information dissemination requires management support for both periodic and aperiodic data updates, while the literature of real-time databases only talks about periodic data updates. In real-time databases the data updates are issued by either active data sources, while both [22] and this paper considers the two types of sources. However, the [22] assumes that only periodic updates have deadlines. Furthermore, the proposed system considers two types of data consumers' requests: queries for the current data and subscriptions for context notifications, while the other approaches consider only data queries. System requirements and objectives are different for each data management system. The analysed context dissemination system cares about both the minimization of the missed deadlines of the delivered updates and the maximization of the freshness of the stored data. This is the reason why the EDF fails, contrary to most of the other approaches that aim to minimize the deadlines of the user transactions. Another aspect of systems' differentiation is the priority assignment between data updates requests and data consumer requests. As it is discussed in [14], it is up to the system designer to select the scheme that best fits. For example, the [17] assumes that data updates never miss their deadlines; the [22] handles both types based on their deadline; this paper assumes that the load imposed by data consumers' requests is minimal and assigns higher priority to them.

# 6 CONCLUSIONS

This paper has dealt with the scheduling issues arising in field of context information management. The problem of determining the handling order of the information update requests issued by the context sources is formulated given the resource and the timing constraints. Both active (providing values spontaneously) and passive (returning values upon request) types of context sources are considered, in combination with periodic and aperiodic context information generation. Context consumer requests are distinguished in queries requesting a specific context value and subscriptions for context value notifications. The problem's objective is to prioritise the requests in order to maximize data freshness and satisfy the timing constraints considering firm deadlines and non-pre-emptive order. In order to achieve this, the dynamic scheduling algorithm called IPU has been proposed. The IPU takes into account the urgency (describing the timing requirement for serving the request), the importance (describing the profit of serving the request) and the popularity (describing the demand of this request) so that more requests are answered successfully, while it also exhibits fairness through trying to scatter any deadline misses across the different context sources. The evaluation of the proposed algorithm against static and dynamic state-of-the-art algorithms, such as First In First Out, Earliest Deadline First, Rate Monotonic, showed that IPU outperforms the traditional approaches. The most important performance metrics used for the assessment are the fraction of context notifications answered successfully and the fraction of context queries answered with fresh context values. The results of this work are not only applicable to context-aware systems, but also to applications that require real-time data collection such as network management, stock trading, air traffic control and medical applications.

## REFERENCES


[1] M. Satyanarayanan, Pervasive Computing: Vision and Challenges, IEEE Personal Communications Magazine, Vol. 8, No. 4, pp. 10–17(Aug. 2001).

[2] S. Xynogalas, I. Roussaki, M. Chantzara, and M. Anagnostou, Context Management in Virtual Home Environment Systems, Journal of Circuits, Systems, and Computers, Vol. 13, No. 2 (Apr. 2004).

[3] S. Xynogalas, M. Chantzara, I. Sygkouna, S. Vrontis, I. Roussaki and M. Anagnostou, Context Management for the Provision of Adaptive Services to Roaming Users, IEEE Wireless Communications, Vol. 11, N. 2, pp. 40-47 (Apr. 2004).

[4] J. Stankovic, Misconceptions About Real Time Computing, IEEE Computer, Vol. 21, No. 10 (Oct. 1988).

[5] J. Stankovic, S. Son and J. Hansson, Misconceptions About Real-Time Databases, Computer, Vol. 32, No. 6, pp. 29-36 (June 1999).

[6] A. Bestavros, S. Son and K. Lin, Real-Time Database Systems: Issues and Applications, Kluwer Academic Publishers, Norwell, MA, (1997).

[7] G. Ozsoyoglu and R.Snodgrass, Temporal and Real-Time Databases: A Survey, IEEE Transactions on Knowledge and Data Engineering, Vol. 7, No. 4, pp. 513-532 (Aug. 1995).

[8] A. Dey, Understanding and using context, Personal and Ubiquitous Computing Journal, Vol. 5, No. 1, pp. 4-7 (Febr. 2001).

[9] CONTEXT: Active Creation, Delivery and Management of efficient Context Aware Services, IST-2001-38142-CONTEXT, http://context.upc.es.

[10] I. Sygkouna, M. Chantzara, S. Vrontis, S., Xynogalas, M. Anagnostou and E. Sykas, Seamless networking and QoS provisioning for context-cware services in heterogeneous environments, LNCS Proc. of the 2nd International Workshop on Mobility Aware Technologies and Applications (MATA05) (Oct. 2005).

[11] M. Chantzara, M. Anagnostou and E. Sykas, Designing a Quality-Aware Discovery Mechanism for Acquiring Context Information, Proc. of the IEEE 20th International Conference on Advanced Information Networking and Applications (AINA06), Vol. 1, pp. 211-216 (Apr. 2006).

[12] J. Haritsa, M. Carey and M. Livny, Value-Based Scheduling in Real-Time Database Systems, VLDB Journal, Vol. 2, pp. 117-152 (1993).

[13] K. Ramamritham, S. Son and L. Dipippo, Real-Time Databases and Data Services, Real-Time Systems Journal, Vol. 28, No. 2-3, pp. 179-215 (Nov./ Dec. 2004).

[14] L. Sha, T. Abdelzaher, K-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky and A. K. Mok, Real Time Scheduling Theory: A Historical Perspective, Real-Time Systems Journal, Vol. 28, No, 2-3, pp. 101-155 (Nov./Dec. 2004).

[15] B. Adelberg, H. Garcia-Molina and B. Kao, Applying update streams in a soft real-time database system, ACM SIGMOD Record, Vol. 24, No. 2, pp. 245-256 (May 1995).

[16] X. Song and J.W.S. Liu, Maintaining Temporal Consistency:Pessimistic versus Optimistic Concurrency Control, IEEE Transactions on Knowledge and Data Engineering, Vol. 7, No. 5, pp. 786-796 (Oct. 1995).

[17] M. Xiong, K. Ramamritham, J. Stankovic, D. Towsley and R. Sivasankaran, Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics, IEEE Transactions on Knowledge and Data Engineering, Vol. 14, No. 5, pp.1155-1166(Sept. 2002).

[18] K. Kang, S. Son and J. Stankovic, Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases, IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 10 (Oct. 2004).

[19] K. Kang, S. Son and J. Stankovic, Differentiated Real-Time Data Services for E-Commerce Applications, E- Commerce Research, Special Issue on Business Process Integration and E-Commerce Infrastructure, Kluwer Academic Publishers, Vol. 3, No. 1-2, pp 113-142 (Jan./Apr. 2003).

[20] M. Amirijoo, J. Hansson, S. Son, Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations, IEEE Transactions on Computers, Vol. 55, No. 3, pp. 304-319 (March 2006).

[21] Qi Han and N. Venkatasubramanian, Information Collection Services for QoS-aware Mobile Applications, IEEE Transactions on Mobile Computing, Vol. 5, No. 5, pp. 518- 535 (May 2006).

[22] Qi Han and N. Venkatasubramanian, Addressing timeliness/accuracy/cost tradeoffs in information collection for dynamic environments, Proc. of the IEEE Real-Time Systems Symposium 2003, pp.108-117 (Dec. 2003).