지능정보연구 제17권 제1호 2011년 3월(pp.01~16)

Queuing Time Computation Algorithm for Sensor Data Processing in Real-time Ubiquitous Environment

Kyung Woo Kang
Division of Information and Communication,
Baekseok University
(kwkang@bu.ac.kr)

Ohbyung Kwon
College of Management,
Kyung Hee University
(obkwon@khu.ac.kr)

The real-time ubiquitous environment is required to be able to process a series of sensor data within limited time. The whole sensor data processing consists of several phases : getting data out of sensor, acquiring context and responding to users. The ubiquitous computing middleware is aware of the context using the input sensor data and a series of data from database or knowledge-base, makes a decision suitable for the context and shows a response according to the decision. When the real-time ubiquitous environment gets a set of sensor data as its input, it needs to be able to estimate the delay-time of the sensor data considering the available resource and the priority of it for scheduling a series of sensor data. Also the sensor data of higher priority can stop the processing of proceeding sensor data. The research field for such a decision making is not yet vibrant. In this paper, we propose a queuing time computation algorithm for sensor data processing in real-time ubiquitous environment.

Received : December 25, 2010 Revision : January 06, 2011 Accepted : January 09, 2011
Type of Submission : English Fast-track Corresponding author : Ohbyung Kwon

1. Introduction

Ubiquitous computing environment becomes applied to a variety of service areas such as logistics, asset management and commerce. Ubiquitous computing environment aims to be also available in a wider range of physical domain such as u-cities. However, this enlargement must require resolving scalability issues. Considering the ubiquitous computing vision characterized by mobility of people, anywhere connected, distributed context data processing, heterogeneous devices, and individualized response, scalability and timely response have been regarded as crucial. In particular, the essential part of scalability lies in endurably processing huge sensor data from many sensors in limited time. Data processing becomes more intensive if context aware process needs

non-sensor data such as user profile and web source to be jointly used with sensor data in real-time.

This paper focuses on scheduling problem for scalable sensor data processing in real-time ubiquitous environment. In general, real-time ubiquitous environment consists of receiving sensor data from sensory networks, retrieving non-sensor data from database or any other data sources, acquiring context from sensor data and non-sensor data, and returning context data to user interface. These steps may vary subject to the characteristics of the domain. For example, sensor data may be obtained form heterogeneous sensors: location, temperature, humidity, illumination, or activity. Then the sensor data may be temporarily stored in a specific repository or working memory, in case of database and streaming method, respectively. Reading the stored sensor data is performed by more than one application programs which are connected to database management system or Data Stream Management System (DSMS). Once the data are read, one can estimate context information which is useful to provide a set of services to the users. At this time, inference engines may involve learning algorithms such as Support Vector Machine (SVM), Markov analysis and time series analysis for real-time service.

Not a few portions of ubiquitous computing environments must have real-time features. Correspondingly, at least three conditions required by real-time environment are needed in the realization of real-time ubiquitous service. First, required time of service from reading sensor data to make context information should be considered. The read sensor data are continuously cumulated in a data queue for service, even though available resources for data processing are limited. More productive data processing and queue administration methods are highly needed. Second, prioritizing sensor data processing tasks stacked in queue should be considered. This is crucial for increasing the quality of service by declining service mismatch issues. To do so, estimated time of service and queuing time of the tasks must be identified to assign more resources with the tasks of higher priority. Accordingly, having estimated waiting time of sensor data of lower priority is also useful in identifying service execution time and notifying it to the users. Third, interval time from sensor data acquisition to service execution should not exceed the expected service execution time. If a data processing task of normal or higher priority may not be accomplished in a required time and the priority of the next task is lower than the current task, then the next task may be deleted to maintain the overall quality of service.

However, even though these conditions are crucial in developing real-time ubiquitous computing environment, remarkable solutions that cope with the conditions are still rare. The dearth of these considerations partially results in the limited realization of real-time ubiquitous services for very small application domains. Sensor data processing methods really matter for larger-sized service domain (Nam et al., 2008). In

particular, estimating queuing time of stacked sensor data is essential to propose more productive sensor data processing methods. This is more critical if one or more inference engines such as ontology engine and rule-based engine are considered in a real-time ubiquitous computing environment (Hatala et al., 2005).

Hence, the purpose of this paper is to propose an algorithm to compute queuing time when a ubiquitous middleware processes sensor data and at the same time newly coming sensor data. Then the optimal queuing time is utilized to make a scheduling decision for real-time ubiquitous computing environment. The remainder of this paper is organized as follows. Some illustrative ubiquitous scenarios which aim to understand the nature of real-time ubiquitous and legacy sensor data processing methods for context-aware services are described in section 2 and 3, respectively. In section 4, the proposed sensor data processing method which computes the queuing time for context-awareness is shown. To show the feasibility of the proposed idea, the design and conducted results of a computational experiment are delineated in the next section. Finally, in section 6, concluding remarks with future research issues are described.

2. Real-Time Ubiquitous Computing Environment Scenarios

In this section, two examples are illustrated to show the nature of real-time ubiquitous computing environments, and also the reason why the research of queuing time computation for sensor data processing is doable. The first example is customized advertisement in the shopping mall. Suppose that some products are promoted on a customer's device when the customer is on the way into a mall from parking lot. The contents for individualized promotion may have nothing to do with age or sex of the customers on the way. In general, if the contents may be about the most popular products or seasonal products, then these promotions may not impress the customers favorably. The advertisement method can be changed when it comes to real-time ubiquitous computing environment setting. Not a few physical sensors are detecting a customer who is arriving at parking lot and the sends the raw data to the inference engine. The engine can get user profile such as sex or age of the customers. With the user profile, the context-aware process gets the list of products that may be suitable to the customers from database or knowledge base. The promotion contents of the products are shown through the display device on the way of the customers. Before the customer passes the display device, the ubiquitous middleware should be ready to show the promotions. It means that the context management middleware should finish the whole process of gathering sensor data, inference, retrieving promotion contents from database and displaying the contents. If it cannot deal with the sensor data of all customers within the time-limit because of the number of them, some of the sensor data should be thrown away, just because it is useless to play the contents or show-

ing ads after the customers pass over the displaying device. Another issue on sensor data is the prioritizing sensor data streams. If sensor data of higher priority come in, the jobs of lower priority should be paused until the higher priority data finishes being processed. For example, suppose that a very important customer comes in the parking lot and can then be recognized by a sensor. The ubiquitous middleware would better prepare the promotion contents which fit more to the customer's dynamic preference. Preparing these contents should get higher priority because the customer will more likely to accept the promotion. Therefore, if the middleware is dealing with several precedent sensor data, the middleware will pause the processing and take the newly coming sensor data of higher priority. If the stopped job of lower priority is estimated to wait later than the service time limit, the job had better be thrown away for global service optimization.

The second scenario refers to scheduling efficient routing in amusement park. In general, visitors are positive to get served for optimal routing reflecting current context such as weather, location, crowd, special events and their preferences. Depending on the context, the visitors select the next attraction to visit. The ubiquitous middleware could support the visitors by providing them with the information how long they have to wait on the line and get served. To do so, the ubiquitous system may encourage the visitor to submit the list of favored attractions at the entrance and be given an ID tag. In the park, their tags would be detected by many optical or IR sensors wherever they enter. From the database, the ubiquitous middleware could retrieve the information of the visitor's preference. For example, current congestion of the rides could be used for recommending the rides. Let's supposed that there are sufficient number of display devices and sensors all over the park. The sensors detect the IDs of visitors in the vicinity and send them to the middleware. The middleware infers the direction of their progress and get the list of favored rides at the same time. The current congestion of the rides would be shown on the displayer ahead. Then based on the information, the visitors can make a decision where to go as the next ride device. Note that whole process should be finished within a specific time limit. The time limit refers to how long time it takes for the visitor to pass the display device. The time from detecting an ID to displaying the information should be within the time limit. Otherwise, the process for the ID had better be thrown. In this example, we should consider the priority of the visitors. If they would purchase the express tickets, they will be assigned to have higher priority.

These two scenarios have some commonalities that every sensor data should be processed in real time and be shown at the right time. If they cannot be processed in real time and at the right time, the data had better be thrown. The sensor data of higher priority make the progress of precedent sensor data be stopped or abandoned. However, sophisticated methods to estimate the queuing time that is valuable in making a decision if the data processing can be finished within

a required time have been so far rare. Hence, estimating the queuing time of the successive sensor data under the process of precedent sensor data is very useful for the scheduler of real-time ubiquitous computing environment.

3. Related Works

Context information is a set of data which come from sensors or sensory network that surround the user's space. Context information is useful to make an individual context-aware service that enables natural interface. To do so, huge sensor data need to be summarized, interpreted, reorganized or reasoned according to the service objectives. To be actually used in the real-time ubiquitous computing environment, sensor data processing methods must deal with scalability issues. To accomplish this issue, first, sensor data filtering method can be used. Using query commands or pre-defined rules are executed whenever sensor data stream arrives at the context-aware subsystem (Musolesi, 2008). Second, machine learning algorithms such as data mining method can be applied to identify and summarize sensor data set to provide condensed yet valuable context data set (Hastie et al., 2001). For instance, in case we use smart sensors, the machine learning algorithm can be used when the smart sensor or cluster head can preprocess the huge volume of sensor data before sending to central data processing unit. This will enhance scalability and at the same time security concerns. In addition, Random Decrement Technique (RDT), which reflects the space's dynamic features in choosing a sensor data gathering strategy, can apply this preprocessing approach (Sim et al., 2011). Next, context prediction method can be considered to accomplish this scalable sensor data processing method. Based on previous context data history, context prediction method can identify estimated context data value before the event really happens. The context prediction method is more scalable than any other sensor data processing methods, even though the estimation performance in terms of accuracy might not be superior to the competing non-prediction methods. Hence, context prediction method is suitable in developing scalable and less-elaborate real-time ubiquitous environment such as mobile shopping domain, rather than healthcare or security preserving domains. DSMS is another technology that can consider context-awareness in scalable sensor data processing method. DSMS differentiates its way of sensor data management with DBMS in that DSMS can deal with continuously arriving data stream. This capability makes it possible to be applied in network monitoring of sensor network (Galob and Ozsu, 2003). STREAM, developed by Stanford University, is a representative DSMS tool has CQL (Continuous Query Language), which is able to perform query commands in a continuous manner (Arasu et al., 2004).

The essential part of setting up optimal scheduling for sensor data processing is to estimate queuing time after arriving sensor data from sensory network. Accordingly, algorithm for queuing time estimation will be crucial for the sche-

duling. The central unit of queuing time estimation is prioritizing sets of sensor data. Prioritizing sets of sensor data are generated in difference time frames is another crucial task for user-acceptable ubiquitous services in terms of timeliness. For example, RM (Rate Monotonic) performs data processing based on the meta-knowledge that the shorter the cycle time to process a task, the higher should the task processing priority. On the other hand, EDF (Earliest Deadline First) algorithm increases the priority of the task processing when the time to finish is approaching (Liu, 2000; Lee et al., 2010).

4. Queuing Time Computation for Sensor Data Processing

4.1 Target

For computing the queuing time in real-time ubiquitous computing environment, sensor data processing is defined as follows: The ubiquitous middleware is assumed to consist of n phases which are denoted by P_1, P_2, P_3, ⋯, P_n. The number n varies according to the application of ubiquitous environment. Each sensor data is written in S. In the same way as the series of phases, a sequence of sensor data can be denoted by S_1, S_2, S_3, ⋯, S_m in the arrival order. <Table 1> shows two sensor data S_1, S_2 where S_1, S_2 are called precedent, and successive, sensor data, respectively. Let us suppose there is a constraint that any phase is not able to process more than one sensor data simultaneously. However, if ubiquitous middleware operates in multiple phases, it is possible to operate multiple processing of sensor data at the same time. <Table 1> illustrates the expected processing time of each sensor data in the phases. In order to increase the effectiveness of ubiquitous middleware, we need to set the minimum waiting time of the successive sensor data in relation to the precedent sensor data, and the setting of waiting time is calculated according to the schedule programmed in the ubiquitous middleware. The waiting time and latency are important factors to measure the performance of real-time system (Bumbalek et al., 2010; Schurgers et al., 2002). Furthermore, these times could cause the service loss when service time is too long from when the service is requested. The expected processing time of each sensor data in each phase could be estimated with the quantity of data and forecasted loading. The quantity of data can be computed with the amount of data in queue, the transition of the amount and the slope of the transition. The load of the ubiquitous middleware can be forecasted with several methods but the forecasting is out of the scope of this research.

<Table 1> The Expected Time for Processing Sensor Data in each Phase

Sensor data	Time					Order
	P_1	P_2	P_3	⋯	P_n	
S_1	t_{11}	t_{12}	t_{13}	⋯	t_{1n}	1
S_2	t_{21}	t_{22}	t_{23}	⋯	t_{2n}	2

The successive sensor data S_2 should wait

for the precedent sensor data S_1 during following time;

$$D = \min_{1 \le j \le n}(\{d_j | \sum_{1 \le i \le j} t_{1i} = d_j + \sum_{1 \le i \le j-1} t_{2i}\}$$

S_2 begins to show the result at the response time R, which is acquired from adding the waiting time to total processing time.

$$R = D + \sum_{1 \le i \le n} t_{2i}$$

If the response time of S_2 exceeds the limited time that real-time ubiquitous environment sets, then S_2 should be thrown away from queue and the successive sensor data of S_2 should be selected for computing the response time of it. In order to minimize the response time, the waiting time should be minimized. In this paper, an algorithm for computing minimum waiting time is proposed for minimizing the response time in the real-time ubiquitous environment.

4.2 Rules

The minimum time taken for *i-th* sensor data S_i to finish *k-th* phase P_k can be defined as following;

$$T_i(k) = \sum_{1 \le j \le k} t_{ij}$$

When *i-th* sensor data S_i is precedent and *j-th* sensor data S_j is successive, the time for which S_j should wait S_i finishing *k-th* phase P_k is $T_{ij}(k)$.

$$T_{ij}(k) = T_i(k) - T_j(k-1) \text{ where } T_j(0) = 0$$

When these formulas are applied to whole phases, the waiting time D can be got as following;

$$D = \max_{1 \le k \le n} T_{ij}(k)$$

D is the waiting time of the successive sensor data under the condition that it is not allowed to pass the precedent sensor data. If the precedent sensor data should be processed on a phase but the successive sensor data does not have to be done on the same phase, the successive one could be allowed to pass the precedent one on that phase. This fact can induce some rules as following

Rule 1: Sequential Processing Rule

Condition : ($t_{ik} \neq 0$ and $t_{jk} \neq 0$) and ($T_{ij}(k) > 0$ and $T_{ji}(k) > 0$) for $1 \le k \le n$

- If S_i is the precedent sensor data, S_j should wait for $T_{ij}(k)$.
- If S_j is the precedent sensor data, S_i should wait for $T_{ji}(k)$.

If two sensor data arrive at the same time, these sensor data should be processed sequentially. The precedent one is eligible to be processed prior to the successive one and the successive one should wait for the waiting time if the successive one does not have higher priority. The waiting time can computed using Rule 1.

Rule 2 : Passing Rule

Condition : ($t_{ik} \neq 0$ and $t_{jk} \neq 0$) and ($T_{ij}(k) < 0$ or $T_{ji}(k) < 0$) for $1 \le k \le n$

- If $T_{ij}(k) \leq 0$ and S_i is the precedent sensor data, S_j does not need to wait.
- If $T_{ji}(k) \leq 0$ and S_j is the precedent sensor data, S_i does not need to wait.

This condition means that the precedent one is being processed at (*k-1)*-th phase and the successive one does not need to be processed at the same phase. Therefore the successive one is eligible to enter *k*-th phase prior to the precedent one.

Rule 3 : Independent Processing Rule

Condition : $(t_{ik} = 0$ or $t_{jk} = 0)$ for $1 \leq k \leq n$

- One sensor data does not need to wait for the other one since they do not share *k*-th phase P_k.

This condition says indicates that at least one sensor data does not need to be processed at *k*-th phase, that is, two sensor data do not run into each other at the phase. Therefore two sensor data do not disturb each other.

The waiting time D can be derived using Rule 1, Rule 2 and Rule 3 as follows :

$$D = \min(\{d|T_{ij}(k)\text{-}d \leq 0 \text{ or } T_{ji}(k)+d \leq 0 \text{ for } 1 \leq k \leq n \text{ where } t_{ik} \neq 0,\ t_{jk} \neq 0\}$$

Moreover, following fact can be discovered based on three rules. If ($t_{jk} = 0$ and $T_{ij}(k\text{-}1)\text{-}D \leq 0$ and $T_{ji}(k+1)\text{-}D \leq 0$ for $1 \leq k \leq n$) is true, the successive sensor data S_j could pass the precedent sensor data S_i at the phase P_k. Likewise, if ($t_{ik} = 0$ and $T_{ji}(k\text{-}1)\text{-}D \leq 0$ and $T_{ij}(k+1)\text{-}D \leq 0$ for $1 \leq k \leq n$) is true, the successive sensor data S_i could pass the precedent sensor data S_j at the phase P_k.

To induce the three rules, let's suppose that two sensor data arrive the queue at the same time and queuing order is assigned arbitrarily. Later on, the waiting time will be defined without the assumption. The arrival times of the sensor data S_i, S_j are written in a_i, a_j where $a_i \neq a_j$. The interval time between the two arrival events is either $a_i\text{-}a_j$ or $a_j\text{-}a_i$. If S_j is a successive one, the new waiting time $T'_{ij}(k)$ for which S_j should wait before *k*-th phase P_k can be defined as follows:

$$T'_{ij}(k) = T_i(k)+a_i\text{-}T_j(k\text{-}1)\text{-}a_j = T_{ij}(k)+a_i\text{-}a_j$$

Rule 1, Rule 2 and Rule 3 also should be modified as $T'_{ij}(k)$ from $T_{ij}(k)$. The modified rules can be applied to obtain the new waiting time D′

$$D' = \min(\{d|T'_{ij}(k)\text{-}d \leq 0 \text{ or } T'_{ji}(k)+d \leq 0 \text{ for } 1 \leq k \leq n \text{ where } t_{ik} \neq 0,\ t_{jk} \neq 0\}$$

$T'_{ij}(k)$ and D′ help to compute the waiting time of successive sensor data if the number of precedent sensor data is greater than 1. If there are multiple sensor data in the queue and some of them under the process of ubiquitous middleware, the waiting time of a new arriving sensor data can be computed with the processing time of all precedent sensor data. In this paper, Algorithm 1 is proposed for the computation.

Algorithm 1 : Computation of waiting time for the precedent sensor data

Input : Processing time of precedent sensor data

(S_1, S_2, ⋯ , S_m) and successive sensor data (S_l)

Output: Waiting time (D'_l) of the successive sensor data (S_l)

Assumption: Ubiquitous middleware consists of n phases (P_1, P_2, ⋯ , P_n).

1. $D_l = a_l$
2. Repeat
3. For $i = 1$ to m
4. If $a_i \leq a_l$ Then
5. $T'_i(k) = \sum_{l \leq j \leq k} t_{ij}$ for $1 \leq k \leq n$
6. $T'_l(k) = \sum_{l \leq j \leq k} t_{ij} + a_l - a_i$ for $1 \leq k \leq n$
7. Else
8. $T'_i(k) = \sum_{l \leq j \leq k} t_{ij} + a_i - a_l$ for $1 \leq k \leq$ n
9. $T'_l(k) = \sum_{l \leq j \leq k} t_{ij}$ for $1 \leq k \leq$ n
10. End If
11. $T'_{il}(k) = T'_i(k) - T'_l(k-1)$ and $T'_{li}(k)$ $= T'_l(k) - T'_i(k-1)$ where $T'_i(0) = 0$ for $1 \leq k \leq n$
12. $D' = \min(\{d | T'_{il}(k) - d \leq 0$ or $T'_{li}(k) + d \leq 0$ for $1 \leq k \leq n$ where $t_{ik} \neq 0$, $t_{jk} \neq 0\}$
13. $a_j = a_j + D'$
14. End For
15. Until no change of a_j
16. $D_l = a_l - D_l$

Note that for-loop begins at line 3. In the loop, the i-th iteration computes the waiting time which S_l waits for during the process of a precedent sensor data S_i. The iteration begins with the first precedent sensor data S_1 since the sequence of all precedent sensor data is arranged in the arrival order, that is, the order says the FCFS (First-Come First-Service) priority. One iteration may not be sufficient to compute the final waiting time because all precedent sensor data may influence the successive sensor data. Therefore once the waiting time of the successive sensor data is changed by one precedent sensor data, it should be checked whether the waiting time might be also changed by the other precedent sensor data. The work should continue until the waiting time is not changed any longer. In the Algorithm 1, the Repeat-Until is used for the work. The loop is terminated if the new waiting time is not influenced by any precedent sensor data. New waiting time becomes the final waiting time of the successive sensor data.

5. Experiments

In Section 3, we proposed several formulas for computing the queuing time of the sensor data, that is, how long sensor data should wait before being processed by real-time ubiquitous computing environment. The queuing time is the minimum waiting time under the given constraints and should not influence precedent sensor data proceeding. In order to evaluate the accuracy of our formulas and the proposed algorithm, we experimented with typical hypothetical data under several situations.

For the experiment, we have implemented and compared the computation schema described for queuing time in Section 3. In the first experiment, the waiting time between two sensor data is computed. The second experiment shows the waiting time among multiple sensor data.

The first experiment is divided into four kinds of schema. There are two criteria for classification of the schema. The first criterion is whether successive sensor data is allowed to pass precedent sensor data or not. Second one is whether the sensor data consider the priority of sensor data. If the successive sensor data have higher priority than the precedent, the precedent should stop for letting the successive pass. <Table 2> shows 4 kinds of schemes.

It is supposed that the expected processing time of every sensor data can be forecasted for this experiment. <Table 3> shows four sets of experimental data each of which consists of the expected processing time for a pair of sensor data. S_1 is the precedent and S_2 is the successive. In the experiment considering priority, we assume that S_2 has the higher priority than S_1.

<Table 4> shows the delay time which the successive sensor data or the sensor data of the lower priority should wait for when the data sets of <Table 3> are applied to each scheme.

The schemes of allowing pass (Scheme 2, Scheme 4) generate the shorter waiting time than the Schemes of no pass (Scheme 1, Scheme 3) on the same condition respectively. The schemes of stopping for higher priority (Scheme 3, Scheme 4) show better result than the schema of non-stop

<Table 2> Kinds of Schemes

	Non Stop	Stop for Higher Priority
No Pass	Scheme 1 : The first-come sensor data always is processed first in all phases. Every sensor data can proceed without a stop.	Scheme 2 : The sensor data of higher priority is allowed to proceed first if two sensor data should be processed in the same phase. The sensor data of lower priority should stop for the higher.
Pass	Scheme 3 : The first-come sensor data is selected first by ubiquitous middleware. Then the sensor data have other sensor data pass on condition of non-stop.	Scheme 4 : The sensor data of lower priority should stop for the higher to proceed. Pass is allowed between sensor data in all phases on condition of no stop.

<Table 3> Experiment Data

Sensor Data		Arrival Time (ms)	The processing Time(ms)					
			P_1	P_2	P_3	P_4	P_5	P_6
Set 1	S_1	0	100	100	300	100	0	300
	S_2	10	100	200	0	200	300	0
Set 2	S_1	0	100	100	200	100	300	0
	S_2	20	0	200	0	100	0	300
Set 3	S_1	0	100	100	0	200	0	100
	S_2	30	0	100	200	0	300	100
Set 4	S_1	0	0	200	200	300	0	100
	S_2	40	100	0	200	0	300	100

(Scheme 1, Scheme 2) but not always. The waiting time Scheme 3 makes with Set 3 is worse than Scheme 1. With most other sets, the results of Scheme 3 and Scheme 4 are better than Scheme 1 and Scheme 2. <Table 4> shows that ubiquitous middleware had better compute the waiting time with Scheme 4(stopping for higher priority and allowing pass).

We performed the second experiment for computing the waiting time when multiple sensor data queue in arrival order. Algorithm 1 is used for computing the waiting time of a successive sensor data for multiple precedent sensor data. <Table 5> shows 8 experimental sensor data with the expected processing time of each phase. The arrival time is the time when each sensor data begin to queue.

The queuing time of each sensor data can be shown in <Table 6> as an experimental result.

In <Table 6>, S_3 should wait for 300ms but S_4 could be processed as soon as it arrives. Since there is no processing time for S_4 at phases P_1 and P_2, S_4 could pass while S_1, S_2 and S_3 are being processed at these phases. S_3 could pass S_1 but could not pass S_2 so it should wait. Likewise, S_6 could pass S_1, S_5 but other sensor data give

<Table 4> Waiting Time(ms)

Scheme \ Data Set	Set 1	Set 2	Set 3	Set 4
Scheme 1	290	480	170	360
Scheme 2	290	280	170	260
Scheme 3	210	120	330	140
Scheme 4	210	120	30	140

<Table 5> Multiple Sensor Data in the Queue

Data	Arrival Time (ms)	Expected Processing Time(ms)					
		P_1	P_2	P_3	P_4	P_5	P_6
S_1	0	500	0	0	100	400	0
S_2	0	0	300	100	100	0	100
S_3	0	0	200	100	200	0	200
S_4	0	0	0	300	100	100	0
S_5	40	100	100	0	200	0	100
S_6	50	0	100	200	0	300	100
S_7	60	0	200	200	300	0	100
S_8	70	100	0	200	0	300	100

<Table 6> Waiting Time

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
0	0	300	0	560	750	840	1130

it a waiting time. If it begins to be processed after the waiting time, it collides with S_1, S_5 at the same phase unluckily. The collision makes the waiting time longer. S_6 should submit to the longer waiting time because it arrives late.

In the previous experiment, the successive sensor data should submit to the longer waiting time in most cases. If the sensor data has a higher priority, it is allowed to be processed first with stopping precedent sensor data. For the experiment of stopping for higher priority, we test Scheme 4 with the test data of <Table 5>. It is assumed that the successive sensor data of higher priority is S_9 and the processing time of it is shown in <Table 7>.

When S_9 arrives, S_1, S_2, S_4 are under process and other sensor data wait in the queue. S_1, S_2, S_4 are stopped and the waiting time of each sensor data should be computed again since S_9 has the higher priority. The new waiting times of all sensor data are shown in <Table 8>. We drop the last sensor data S_8 in the table since the length of queue is assumed to be 8.

S_1, S_4 keep being processed since they will not collide with S_9 until last phase. S_2 meets it at P_2 so S_2 should give the phase to it and wait. The new waiting time of S_2 let it collide with another sensor data. Finally the waiting time of S_2 becomes 200ms. The other queued sensor data should wait for 100ms more for S_9 arriving.

6. Conclusion

This paper proposed an algorithm which computes estimated queuing time of sensor data which are continuously arrived from sensory network under real-time ubiquitous environment. Decreasing the queuing time will more likely to increase the volume of simultaneously processed sensor data and hence improve the real-time ubiquitous environment in terms of reliability. Four different methods were considered to compute the estimated queuing time using experimental data. As expected, we observed the best results in case of allowing prioritization of sensor data. Moreover, based on these findings, we performed a computational experiment which estimates queuing time of newly arriving sensor data by referring the elapsed time to processing preceding sensor data.

<Table 7> Sensor Data of Higher Priority

Data	Arrival Time (ms)	Processing Time(ms)					
		P1	P_2	P3	P4	P5	P6
S_9	100	0	100	0	100	0	100

<Table 8> The New Waiting Times of All Sensor Data(ms)

S_9	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	0	200	400	0	660	850	940

Since legacy studies have assumed small-scale ubiquitous environment with a limited number of sensors, they seldom coped with scalability and time to service issues. However, actual domains to apply the ubiquitous systems are much larger scale such as shopping malls, streets and even u-cities. For example, u-city project, which is an integrated set of multi-domain u-services for many people, should consider tremendous amounts of sensor data at a time generated in considerable number of sensors. Moreover, if the ubiquitous services aim individualized and timely service for the people who are walking and even driving, then these must require scalability free context-aware system. The start point of implementing scalability free context-aware system is a rapid scheduling of processing sensor data. Hence, the results of this paper will contribute to satisfy the necessary condition for scalability free context-aware system in real-time ubiquitous environment. First of all, real-time multi-agent based scheduling can be supported by the algorithm proposed in this paper. Real-time multi-agent based scheduling is a kind of decision making problem which decides the priorities of the tasks of the agents (Garvey and Lesser, 1993). Second, our algorithm can be applied in mobility-conscious scheduling problems which consider fast moving users or mobile sensors. For example, context aware applications embedded in intelligent robots, real-time sensor data processing, tracking mobile object problems using mobile RFID, RTLS (Real Time Location System), and remote monitoring system can be supported by the proposed algorithm (Nam et al., 2008).

References

Arasu, A., B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, *STREAM : The Stanford Data Stream Management System,* Technical Report. Stanford InfoLab, 2004.

Bumbalek, Z., J. Zelenka, and L. Kencl, e-Scribe : Ubiquitous Real-Time Speech Transcription for the Hearing-Impaired, http://www.rdc.cz/download/publications/escribe.pdf.

Galob, L., M. T. Ozsu, "Issues in Data Stream Management", *ACM SIGMOD Record*, Vol. 32, No.2(2003), 5～14.

Garvey, A. and V. Lesser, Design-to-time Real-Time Scheduling, *IEEE Transactions on Systems, Man and Cybernetics*, Vol.23, No.6 (1993).

Hastie, T., R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning : Data Mining, Inference, and Prediction*, Springer, 2001.

Hatala, M., R. Wakkary, L. Kalantari, Rules and ontologies in support of real-time ubiquitous application, *Web Semantics*, (2005), 5～22.

Lee, B., S. Lim, J. Kim, "Scalable real-time monitoring system for ubiquitous smart space", *Information Processing Letters*, Vol.110 (2010), 294～299.

Liu, J. W. S., *Real-Time Systems*, Prentice Hall, 2000.

Musolesi, M., "Real-Time Ubiquitous Urban Sensing and Modelling", *IEEE Internet Computing*, Vol.12, No.4(2008), 12～21.

Nam, M., M. Z. Al-Sabbagh, J. Kim, M. Yoon, C. Lee, and E. Y. Ha, "A Real-Time

Ubiquitous System for Assisted Living : Combined Scheduling of Sensing and Communication for Real-Time Tracking", *IEEE Transactions on Computers*, Vol.57, No.6 (2008), 795～808.

Schurgers, C., V. Tsiatsis, S. Ganeriwal, and M. Srivastava, "Optimizing Sensor Networks in the Energy-Latency-Density Design Space", *IEEE Transaction on Mobile Computer*, Vol. 1, No.1(2002).

Sim, S., J. F. Carbonell-MArquez, B. F. Spencer Jr., and H. Jo, "Decentralized random decrement technique for efficient data aggregation and system identification in wireless smart sensor networks", *Probabilistic Engineering Mechanics*, Vol.26(2011), 81～91.

Abstract

실시간 유비쿼터스 환경에서 센서 데이터 처리를 위한 대기시간 산출 알고리즘

강경우* · 권오병**

실시간 유비쿼터스 환경은 센서로부터 얻어낸 데이터를 기반으로 상황을 인지하고 사용자에게 적절한 반응을 보이기까지 제한된 시간 내에 모든 것을 처리해야 한다. 전체적인 센서 데이터 처리는 센서로부터의 자료 확보, 상황 정보의 획득, 그리고 사용자로의 반응이라고 하는 과정을 거친다. 즉, 유비쿼터스 컴퓨팅 미들웨어는 입력된 센서 자료 및 데이터베이스 또는 지식베이스로부터 일련의 자료들을 활용하여 상황을 인식하며, 그 상황에 적합한 반응을 하게 된다. 그런데 실시간 환경의 특징 상 센서데이터가 들어오면 각 가용 자원들을 검색하고 그 곳에 있는 미들웨어가 데이터를 처리 할 경우 어느 정도의 대기시간이 필요한지를 결정해야 한다. 또한 센서 데이터 처리의 우선순위가 높을 때는 미들웨어가 현재 처리중인 데이터를 언제 처리를 중지하고 얼마나 대기시켜야 하는지도 결정해야 한다. 그러나 이러한 의사결정에 대한 연구는 아직 활발하지 않다. 따라서 본 논문에서는 유비쿼터스 미들웨어가 이미 센서 데이터를 처리하고 있고 동시에 새로운 센서 데이터를 처리해야 할 때 각 작업의 최적 대기시간을 계산하고 스케줄링하는 알고리즘을 제안한다.

Keywords : 실시간 시스템, 유비쿼터스 컴퓨팅, 센서 데이터 처리, 상황인식 시스템

* 백석대학교 정보통신학부 부교수
** 경희대학교 경영대학 교수

저 자 소 개

Kyung Woo Kang

Presently an associate professor at Division of Information and Communication, Baekseok University, South Korea, where he initially joined in 2000. He received the BS degree at Kyung Sung University in 1990, and MS and PhD degree in Computer Science at KAIST in 1992 and 1998, respectively. From 1998 to 2000, he worked for Super Computing Center at ETRI. In addition, he is now a visiting professor at Singapore Nanyang Technology University. His current research interests include Ubiquitous Computing, Grid Computing, and Compiler Theory.

Ohbyung Kwon

Presently a professor at Kyung Hee University, South Korea, where he initially joined in 2004. In addition, he is now working for Department of Information and Decision Systems at San Diego State University as an adjunct professor. In 2002, he joined Institute of Software Research International (ISRI) at Carnegie Mellon University to perform myCampus project on semantic web and context-aware computing. He received the MS and PhD degree in Management Information System at KAIST (Korea Advanced Institute of Science and Technology) in 1990 and 1995, respectively. His current research interests include ubiquitous computing services, agent technology, mobile commerce, context-aware system development, case-based reasoning, and DSS. He has published various papers in leading information system journals such as Decision Support Systems.