Proceedings of the International Conference on Information and Automation, December 15-18, 2005, Colombo, Sri Lanka.

Resource-Aware Ubiquitous Data Stream Querying

Iti Agarwal*, Shonali Krishnaswamy+, and Mohamed Medhat Gaber++
School of Computer Science and Software Engineering
Monash University, Melbourne, Victoria 3145
*Email: iaga1@student.monash.edu.au
+Email: Shonali.Krishnaswamy@infotech.monash.edu.au
++Email: Mohamed.Medhat.Gaber@infotech.monash.edu.au

***Abstract*—This paper proposes and develops a novel, iterative model for resource aware- ubiquitous data stream querying (RA-UDSQ). Our model provides timely results to mobile users at regular time intervals specified by the user, thereby executing continuous stream queries. This model is capable of adapting to high data rates of streams and limited memory resources available on a mobile device while executing continuous queries. We have experimentally compared our resource-aware ubiquitous data stream querying model with data stream querying without the adaptation. Our comparative analysis has shown that our RA-UDSQ can perform better in terms of memory and battery without compromising on accuracy of results when data rates are high and available resources are critically low.**

I. Introduction

In the recent past, applications such as wireless sensor networks, click streams, web page visits [2] have emerged that produce data in the form of rapid flowing and unbounded streams. The data arriving in these streams are "digitally encoded" and are continuous in nature [GKZ04b] [9]. The continuous nature of these data streams makes it challenging to store and manipulate such data formats in traditional databases for future analysis. Therefore, systems such as STREAM [2] and Aurora [1] have been designed for managing data streams. These systems are called Data Stream Management Systems (DSMS). The main challenge faced by these data stream management systems is to design and implement algorithms that are capable of performing the desired operation on the data streams in a "single pass", given the infeasibility of persistent storage of data streams.

Recently, efforts have been made in performing data stream mining in a ubiquitous environment and thus, leading to ubiquitous data stream mining. This extension has been made possible due to the improvement in the wireless communications and increase in the usage of mobile devices. Many algorithms have been developed to perform ubiquitous data stream mining (UDM) on mobile devices by adapting to high data rates and coping with various resource constraints such as available memory, battery and less computational capabilities. These algorithms are useful for the mobile users in decision making by performing fast data stream analysis in wireless sensor networks and discovering the hidden patterns and behaviors that exist between data items of rapid streams.

Ubiquitous data streams processing capabilities has been extended to only one direction till now in the form of UDM. The other principal form of data stream processing is querying which has not been focused upon in applying it in the ubiquitous environments. Although, ubiquitous data stream querying has potential benefits supplementary to UDM such as querying data streams using a known threshold that has been discovered by UDM and providing results to the mobile users at regular time intervals.

To perform ubiquitous data stream mining in resource constrained environment such as on tablets, mobile phone and PDA's, a concept called Algorithm Output Granularity (AOG) [4] has been developed to perform resource-aware Ubiquitous Data Stream Mining (RA-UDM) by enabling data stream mining algorithms to adapt to high data rates and resource limited constraints such as available memory and battery life [5].

In this paper, we propose that supplementary to mining, ubiquitous data stream querying can be performed on wearable and handheld computing devices to monitor data streams. Ubiquitous data stream querying can support and enhance UDM applications as follows:

- Monitoring the number of items a specific stock price goes above a specified threshold and informing a mobile stock broker at regular intervals.
- Counting the number of heavy vehicles that drive at a specific speed through a given intersection every hour.
- Monitoring the ticket availability of the flights that goes to specified destination and informing a mobile user at regular time intervals.

These above listed applications and the concept of adaptation of a stream process to available resources provides the motivations to develop a resource aware ubiquitous data stream querying model. Thus, in summary:

- Ubiquitous data stream querying is useful in providing important and timely information to the users on their mobile devices. The concept of continuous queries has already been developed for data streams in general. There is significant potential to leverage this work and adapt it to ubiquitous environments.
- Considerable amount of work has been done in the field of ubiquitous data stream mining but no such efforts have been shown in applying data stream querying to ubiquitous environment. Thus, there is potential to supplement UDM through ubiquitous data stream querying.
- The development of AOG concept provides a sound theoretical framework to develop data stream querying algorithms that can run successfully on resource constrained devices as it addresses the issues of high data rates and resource constraints.
- To increase the range of data stream management capabilities in ubiquitous environment provides another motivation for applying data stream querying to ubiquitous environment.

Based on the above, our objective is to propose and develop a resource aware ubiquitous data stream querying algorithm (RA-UDSQ) that can execute continuous queries, and adapt to and cope with the high incoming data rates according to memory availability. The paper is organized as follows. Section 2 presents literature review of data stream management systems. Section 3 proposes our resource-aware ubiquitous data stream querying model (RA-UDSQ). Experimental evaluation is presented in Section 4. Finally the paper is concluded in Section 5.

II. DATA STREAM MANAGEMENT SYSTEMS

Traditional Database Management Systems (DBMS) are capable of managing and manipulating the data that is static in nature and stored in the database. However, there is an emerging focus on applications that require processing of data streams that are generated continuously such as sensor readings [6] [2], visits to the websites [2] and online transactions [6]. This in turn has necessitated development of systems that are capable of handling and processing these continuous and unbounded data streams. Data Stream Management Systems (DSMSs) have been proposed and developed to manage and process data streams [2]. DSMSs face many challenges posed by the high data rates and continuous nature of streams and management of buffers based on arrival of new data taking into account available main memory. We provide a brief review of two such DSMSs: STREAM [2] and Aurora [1].

The DSMS developed at Stanford University is called STanford StREam DatA Manager (STREAM) [2]. This DSMS uses a variant of SQL as its query language with allocations of time stamps. Queries are processed using operators on queues whose partial results can be stored in a synopsis structure for further updates. Algorithms have also been developed that can allocate resources to queries in order to maximize the query result and to perform scheduling in order to reduce the queue sizes for inter-operator queries [10]. The current limitations faced by the STREAM DSMS are the centralization of the system and inefficiency in generation of query plans [10].

The Aurora Data Stream Management System developed by [1] is mainly used by applications for monitoring purposes. This DSMS can query a large number of data streams. It also allows permanent storage of data in two ways:
(i) to store the block of streams that is in a queue on the secondary storage if that block has to be processed later.
(ii) to store the intermediate results of the query execution in the form of synopses on the disk for future reference [1].

The main issue of DSMS is to design and implement algorithms which are capable of reading data streams in one pass, evaluating data items against queries and providing an approximate output based on the data items. Techniques that have been used in such algorithms include sampling, sketching, histograms, wavelets, sliding windows and negative results [2].

Data Stream Querying is a fundamental operation performed by the Data Stream Management Systems on continuous data streams. Querying on data streams can be of two types: one time or continuous**.** A one time query means that the given query will compute on the data stream at the specified time when it was specified and provide the output based on the data items of the stream at that particular time. On the other hand, a continuous query continues to operate on the data arriving in streams and provides the output at specified time intervals [2]. Thus the result of continuous queries is mainly approximate. Such queries can be stored and updated to provide some new results. The queries can also be defined in advance to operate on data streams and are termed as predefined continuous queries. Continuous queries on the data streams pose many challenges that are described in the following [2].

Restricted memory for computations: Querying data streams is computationally hard because the data streams are unbounded and continuous in nature which makes it impossible to store them for the purpose of querying. This results in computational limitations due to memory restrictions.

Less computation time: Another challenge in querying data streams is the computation time as the data rates of streams are very high and multiple passes algorithms for querying data streams would result in compromising the accuracy and appropriateness of results. Therefore, new algorithms are required which can query data streams in one pass.

Results based on approximation: The most important challenge of querying data streams is that we cannot get accurate results as the data is not static. The results of querying data streams are based on approximation at the cost of accuracy loss. Therefore the algorithms should be designed such that the difference between the approximated result and the actual result is minimized.

Some of the techniques reviewed in [2] for providing approximate results include:

Sliding Windows is a conditional timestamp and the query operates only on those data stream items which pertain to that timestamp.

Batch Processing involves computing the query in batches over a specified time.

Sampling involves selecting data items randomly from the stream and evaluating only those items against the query.

Synopsis data structures are a summary of the data streams which are stored in the main memory and used in evaluating the query.

To query data streams and provide approximate results to the users, different data stream query languages have been developed which are explained below.

Readers are referred to [2], [6] and [10] for detailed explication of these stream query languages. The next section details the general steps involved in processing of queries on data streams.

III. RA-UDSQ

We propose and develop a Resource Aware UDSQ model that applies the concepts of adaptation to resources.

a. Algorithm Output Granularity (AOG)

Our proposed Resource-Aware Ubiquitous Data Stream Querying (RA-UDSQ) model is based on the concept of Algorithm Output Granularity (AOG). This concept was proposed by Gaber et al [7] for mining data streams in a resource constrained environment where available memory and battery are major constraints. AOG controls the output generated by the mining algorithm according to the available memory and data rate. It operates on the premise that if memory is low and data rate is high then the mining algorithm will produce less accurate results and merge existing knowledge to cope with these constraints.

In this paper, we contend that AOG provides a sound theoretical framework for performing querying of data streams in a ubiquitous environment as it addresses the issues of limited resources and high data rates. Therefore, we use the concepts of AOG to develop our Resource-Aware Ubiquitous Data Stream Querying (RA- UDSQ) model.

A. Resource-Aware Ubiquitous Data Stream Querying (RA-UDSQ) Model

Data stream querying can either be one time queries that pertain to the current state of data or continuous queries where the data in the streams are queried continuously and the results are provided iteratively at specified time intervals [2]. Our main aim is to perform continuous querying on data streams in the ubiquitous environment. Data stream querying in ubiquitous contexts have to address several issues in addition to the traditional question of high data rates namely lack of computer resources such as memory and battery. Therefore, to maintain the continual operation of querying on data streams in ubiquitous environment, we propose to use AOG concepts of adapting the query process according to both the available memory and high data rates.

Figure 1 illustrates the overall model of our RA-UDSQ algorithm. The five phases of the Resource-Aware UDSQ algorithm are iterative in nature and are described below with its formalization.

Phase 1: Identifying User Constraints

In the first phase of the RA-UDSQ algorithm, the user is asked to provide the query along with either of the two constraints:

- For a continuous query, a user may specify the time duration in which the user wants the result. For example, a user requests for the name of the shares that *freezes,* (i.e. reaches the maximum price possible in a day), to be delivered to him/her every 2 hours.
- The number of results that are required by the user. Consider a scenario in which the user's query requires a notification to be issued for every 50 cars that cross an intersection with a speed limit greater than 80 km. In this case the user constraint is not time bound, but rather depends on the outcome of the query operation.

In this study we limit our focus to the first user constraint which is the time duration. Therefore in our model, the time duration specified by the user is denoted by *D*.

Once the user has specified the time duration constraint, the next phase of the RA-UDSQ will commence.

Phase 2: Optimization

This phase does the following:

(i) Once the time duration is specified in phase 1, the algorithm divides the time duration into a number of time frames. Such a strategy enables us to present the results that are recent and timely to the user while maintaining the user specified duration because in every time frame the new fluctuated data rate is considered and it may be possible that the maximum number of data items arriving in streams during the last frame satisfies the query. This strategy has been explained more clearly with a help of an example in the following discussion. The number of time frames is calculated based on the time duration specified by the user and every time frame is of equal unit time. Thus:

· Let TF be the current time frame.
· Let $N(TF)$ be the total number of time frames.
· Let UTF be the unit time frame. It is calculated using the following formula:

$UTF = D / N(TF)$

(ii) The algorithm determines the current data rate. It performs the sampling process using any of the techniques specified in [2] such as sliding windows and load shedding to optimize the reading rate, i.e., the actual number of data items on which the query is executed. [11] explains that given the time duration at which the user wants the output, the rate can be optimized such that it produces the maximum number of outputs during that time frame. It is formulated in the following manner:

· Let *DR(TF)* be the data rate for the time frame *TF*.

· Let *RR(TF)* be the count of actual number of data items read in the time frame *TF*.

Consider the following illustrative scenario where a user specifies the time duration for results to be delivered to be 30 minutes. Let the data rate be 100 data items/min. Let the output rate also be 100 data items/min and the available memory as 150 blocks where each block can hold up to 10 data items. This would mean that a total of 1500 data items can be stored in the available memory. Now if the algorithm reads the data items at the rate at which they are coming then in 15 minutes the total memory will be occupied but we need to provide the user with the results only after 30 minutes and not before that. As mentioned before, this is done to ensure that user is presented with the most recent and timely results based on the fluctuating data rate and to include the arrival of the recent data items in the results. Consider the case where due to the lack of our optimization strategy, the memory is occupied with the 1500 results in the first 15 minutes. In the last 15 minutes it is quite possible that 3000 data items arrived that held the potential of satisfying the query but those data items have to be completely ignored due to memory unavailability. Thus, the final output in this case that we will be presenting to the user will not demonstrate the current situation but would apply to the past scenario. Therefore, we need to control the rate at which the algorithm should read the data items using the sampling method and execute the query on it such that the memory is utilized for the duration of 30 minutes. This technique is beneficial in obtaining the output based on the recent data items. So in this case only 50% of the data rate will be treated as sampled data items because the calculated sampling rate will be 0.5. (Note: The determination of the sampling rate is presented in phase 4).

Phase 3: Executing the Query Using Blocking Operators

In the third phase, all the sampled data items are processed according to the specific query for providing the output as per the user requirements at regular time intervals. This query uses some of the blocking operators such as the having clause, the group by clause, and the order by clause in order to select the specific data items that satisfy the specified clauses as the output. The blocking operators perform different functions which are explained below:

1. Having Clause: Having clause in the queries is used to compare the values, obtained either individually or through a user defined function, with the specified condition. If the value satisfies the condition then the record is considered as one of the result.

2. Group By Clause: As the name suggests, the group by clause is used to group or assimilate the results of the query based on some attribute.

3. Order By Clause: This clause is used in the query to arrange the results in ascending or descending order based on specified attribute before delivering them to the user. In order to associate the query processing with the adaptation to resources, we maintain the Output Rate (OR) and the Output Rate per Time Frame (OR(TF)) as follows:

· Let *OR* be the output rate. Whenever the sampled record satisfies the query condition, the output rate is incremented by 1.

· Let *OR(TF)* be the count of the total number of results obtained in the time frame *TF*.

For each time frame, the results of the query execution termed as intermediate results, are stored in the memory so that they can be sent to the user as part of the final results delivered at the end of the time duration specified by the user.

Phase 4: Memory Management

This phase is responsible for managing the memory in such a way that it does not get filled before the time duration specified by the user. To perform memory management for each time frame, the estimated output rate is calculated based on three parameters: the current data rate, the total output rate till the current time frame and the total data items read till the current time frame. This is followed by the computation of the available memory. Once the estimated output rate and available memory are computed, the sampling rate is determined for the next time frame. Since the calculation of the sampling rate depends on the resource constraints such as available memory, data rate and output rate, this phase is one of the most important parts of the RA-UDSQ model. We formulate this phase in the following manner:

· Let *EOR(TF)* be the estimated output rate for the current time frame. *EOR(TF)* is calculated using the following equation

$EOR(TF) = [\ [\ \sum(\ OR(TF)) / \sum(RR(TF)\ )\ ] / TF\ ] * DR(TF)$

where *OR(TF)* is the total number of results in the time frame *TF*, *RR(TF)* is the total number of actual data items read in time frame *TF* and *DR(TF)* is the data rate for the time frame *TF* as explained in phases 2 and 3.

· Let *M(TF)* be the available memory for the current time frame.

· Let *SR* be the sampling rate. It is determined by comparing the estimated output rate with the available memory such that

$SR = 1$ when $EOR(TF) < M(TF)$

otherwise, SR is calculated using the following equation:

$SR = M(TF) / EOR(TF)$ *when* $EOR(TF) > M(TF)$

The memory management is an on going process. The memory is managed in such a way that it will not be completely occupied before the time duration gets over. Once the time duration is over and the results are sent to the user and then the results from the memory will be clear in order to commence the next cycle.

Phase 5: Delivering

This is the final phase of the RA-UDSQ model, where all the intermediate results of the query are collated and presented to the user after the time duration specified by the user. In order to determine the completion of time duration, the count of every time frame is kept by calculating the starting time at the start of a new time frame and the ending time at the end of that time frame. The difference is calculated and if it is equal to the unit time, calculated in phase 2, it marks the completion of one time frame. Since continuous queries are intended to execute over an extended period of time, the delivery of results merely represents the end of a cycle of the RA-UDSQ process. This process is iterative and upon delivery of the results a new cycle commences with the optimization phase followed by the other phases of the RA-UDSQ model as shown in Figure 1.

Based on the above explanation and formalization, the RA-UDSQ algorithm has been summarized in Figure 2.

In the following section the implementation of the above proposed RA-UDSQ algorithm is presented along with its evaluation to establish its feasibility and benefits for performing data stream querying on resource constrained devices in a ubiquitous environment.

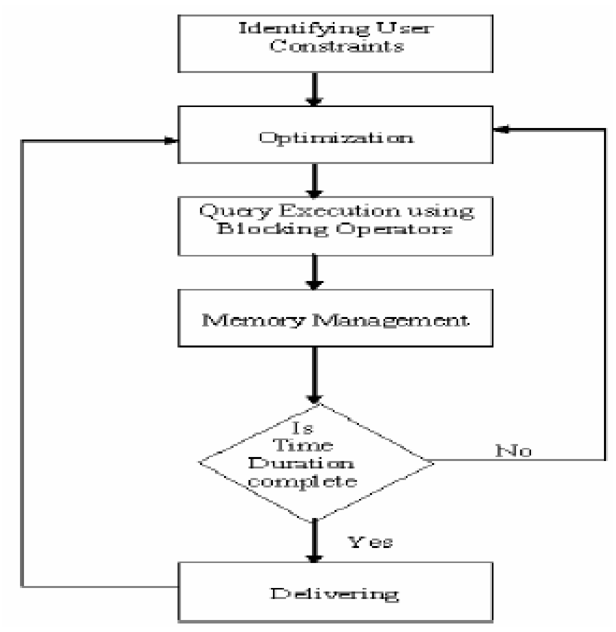

Identifying User Constraints
Optimization
Query Execution using Blocking Operators
Memory Management
Is Time Duration complete
No
Yes
Delivering

Figure 1 Model of RA-UDSQ Algorithm

IV. EXPERIMENTAL RESULTS

The focus of our experimentation has been to investigate the following:

· Memory savings that our approach can facilitate

· The impact of our approach on battery usage

· The impact of our approach on the accuracy which we evaluate in terms of the loss in responses to the query due to adaptation.

Thus, our evaluation strategy has been to execute continuous queries on a PDA. We evaluate the impact of performing this query without the resource aware (RA) aspects of our model (i.e. stand-alone query) and compare the results with executing the same query under the RA-UDSQ model. Both the algorithms were executed under the same experimental conditions and the results were recorded for comparison. Finally, to ensure the same data stream is fed to both queries, we generate and re-play the same data stream for each experimental run.

The experiments were carried out in wireless network environment on a PDA (i.e. HP iPAQ 3970 model). Both RA-UDSQ and UDSQ algorithms were developed using the following configuration: Pentium 4 CPU 1.5 GHz desktop with 256 MB RAM and running Java (TM) 1 Runtime Environment, Standard Edition, Version 1.1.8. The RA-UDSQ and UDSQ algorithms were converted into Jeode and run on the iPAQ with 64 MB, running Microsoft Windows CE version 3.0.9348.

The development of data stream management system to manage and execute random queries is out of our research scope. Therefore, we implement the following continuous query for experimental evaluation:

*Select * from records*
Group by age
Having income>=25000.

The results of the query are grouped together based on the age parameter. The data rates are fluctuated using a random number to simulate the real world ubiquitous environment.

The values of the rest of the parameters, which are data rate, remaining memory, remaining battery and the output rate, are displayed on the *iPAQ* screen during program execution, as shown below in Figure 3, and are recorded in a separate file for the purpose of comparison.

As mentioned earlier, our model is based on time duration specified by the user and this time duration is divided into number of time frames of equal minutes to provide timely and most recent results to the user. In our RA-UDSQ algorithm we divide the time duration into standard number of time frames. In our implementation, we use five time frames. Incase the time duration is less than 5 minutes, then the number of time frames for our algorithm becomes equal to the specified time duration so that unit time frame is of at least 1 minute. The sampling process is performed using sampling method. A random number is generated between 0 and 1 using Java randomize function. This number is compared with the current sampling rate. If the random number is less than the sampling rate then that data item is queried otherwise it is ignored. The following section details the different experiments conducted and our analysis.

We have conducted several experiments to evaluate the performance of our RA-UDSQ algorithm and compared this approach with executing stream queries in a ubiquitous environment without our proposed resource-aware adaptation. These experiments measure three main parameters: available memory, battery consumption and accuracy of the final output. These experiments are discussed as follows:

a. The aim of this experiment is to investigate the impact of RA-UDSQ on memory usage when compared with the memory usage obtained when the same query is executed without resource-aware adaptation.

From hereon, we term the non resource-aware execution of ubiquitous data stream querying as UDSQ. Within each experimental run, we executed the RA-UDSQ and UDSQ algorithms using the same data rates with the same amount of initial available memory for the same time duration. We conducted ten different experimental runs by varying the data rates, initial available memory and time duration for each run. The results of this experiment are presented in Table 1. We record the time duration in minutes, initial physical available memory for program execution calculated in MB and the remaining memory calculated in percentage terms at the end of time duration in case of RA-UDSQ and UDSQ respectively. The last column indicates the memory savings obtained with the execution of RA-UDSQ when compared with UDSQ.

The comparison in memory savings is shown in Table 1. This illustration shows that the remaining memory in case of Resource-Aware UDSQ is always more than the remaining memory in case of without resource aware UDSQ. This clearly indicates that more memory has been saved with the execution of resource aware UDSQ than with the execution of UDSQ algorithm. We also observe from these experiments that whenever the data rates are very high in comparison to the available memory then the difference between the memory savings from RA-UDSQ and UDSQ tends to be correspondingly high. This shows the capability of our approach in coping with high data rates through adaptation.

b. Another set of experiments has been conducted to assess the battery consumption in both cases.

The main aim of this experiment is to investigate the impact on battery consumption with RA-UDSQ as compared to UDSQ with different data rates, initial available memory and different time durations.

After conducting ten similar experiments like part A, we analyzed that in most cases battery consumption is more with RA-UDSQ than with UDSQ. This is due to the extra processing involved in the execution of RA-UDSQ algorithm as compared to the execution of UDSQ algorithm. But a reverse situation can be seen in some of the cases where battery consumption with RA-UDSQ is relatively lower than with UDSQ because in RA-UDSQ, the sampling rate is adjusted accordingly on the basis of three constraints which are available memory, data rate and estimated output rate to read less number of data items instead of all the data items. This scenario is presented in Table 2 showing the values for case 5.

In contrast, in UDSQ the data items are read at the current data rate irrespective of resource constraints. Therefore, the processing of the UDSQ algorithm consumes more battery because a larger number of data items are read and queried and the output is stored in memory by making internal adjustments due to less available memory. Therefore, in critical circumstances resource aware UDSQ proves more beneficial than non-resource-aware UDSQ, while in general RA-UDSQ is seen as having a slightly higher energy consumption profile than UDSQ which is indicated by an average increase of 1.4%.

The two experiments explained above have shown that in case of RA-UDSQ algorithm whenever the resources are critical, the sampling rate is adjusted to cope with high data rates and less available memory. Our premise is that RA-UDSQ compromises on the actual output within tolerable limits.

c. The main aim of this experiment is to examine the impact on the accuracy of the final output produced by RA-UDSQ with respect to UDSQ algorithm.

We term accuracy as the loss in records that are queried to obtain a positive response due to adaptation and its impact on the sampling rate.

Within each experimental run, we executed the RA-UDSQ and UDSQ algorithms using the same data rates with the same amount of initial available memory for the same time duration. We conducted ten different experimental runs by varying the data rates, initial available memory and time duration for each run. Table 3 presents the results of this experiment. We record the total data rates for the corresponding cases, which are obtained by adding the fluctuated data rate over the specified time duration for every case, and data items that positively answer the query in percentage terms produced by RA-UDSQ and UDSQ respectively. The fifth and the last column of the table indicate the difference in final output of RA-UDSQ and UDSQ.

Consider case 5 shown in Table 3, where the total data rate was relatively higher than the available memory, so the sampling rate became equal to 0.34 in order to cope with the high data rates and resource constraints. As a result, less than 50%, i.e., only 2,137 data items were queried. Therefore, the final output at the end of the execution of RA-UDSQ algorithm was reduced to 1,581 as compared to the final output produced by UDSQ algorithm which was equal to 4,723. Thus, the difference in the outputs of RA-UDSQ and UDSQ became as high as 17.04%.

Similarly, in case 7 the difference in final output is 14.92%. This is because unlike UDSQ, in RA-UDSQ the sampling rate was adjusted to 0.26 and 0.36 concurrently to adapt to high data rates and less available memory. Therefore, the output was considerably less in case of RA-UDSQ algorithm when compared with UDSQ algorithm.

On the other hand, the readings shown above in case 8 in table 5b, revealed that whenever the data rate is relatively less than the available memory, there is less need for the adjustment of the sampling rate. In this case the sampling rate was 0.95 and therefore, the difference in final output between both algorithms was of only 4 data items which is equal to 0.02%.

To sum up the results of the experiments, the following points could be concluded:

· Our proposed RA-UDSQ is always more memory efficient than UDSQ.

· Our proposed RA-UDSQ algorithm performs better than UDSQ algorithm in terms of memory savings and battery consumption whenever the data rates are very high and available memory is critically low.

· In normal circumstances when the adaptation based on resource constraints is less by RA-UDSQ, it consumes battery marginally more than UDSQ algorithm because of additional processing involved in the execution of RA-UDSQ algorithm.

· Our approach provides approximate results due to the incorporation of AOG concept. The extent of compromise on the accuracy of results depends on the data rates and available resources. However, this not withstanding, the ability to adapt is of considerable benefit in ubiquitous environments.

We have presented a comprehensive experimental evaluation and analysis of the performance of our proposed RA-UDSQ approach.

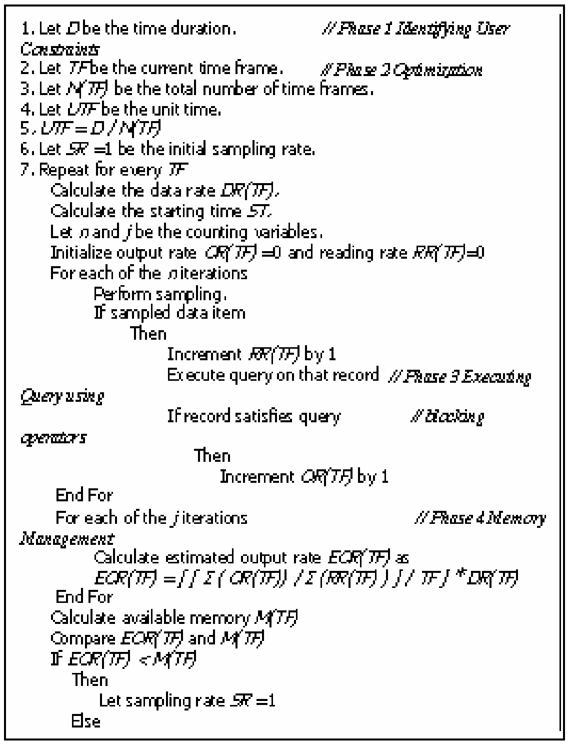
V. CONCLUSION

The emerging focus on data stream applications has necessitated the development of techniques that can analyze and query these data streams. Data stream processing capabilities are being developed for performing mining and querying operations effectively. An increase in wireless communications and improvement in the resources of handheld and mobile devices such as increased computational power and battery life have facilitated the advancement of ubiquitous computing in the recent years. This resulted in the extension of current data stream processing techniques to be performed in ubiquitous context where this challenge is enhanced due to high data rates of streams and resource constraints. There have been many steps taken to realize Ubiquitous Data Stream Mining through the development of light weight mining algorithms [5] that can adapt to high data rates and low available memory to perform mining effectively in the resource constrained environment and UDM systems. However, no efforts have focused on performing data stream querying in a ubiquitous environment. Therefore, this paper has proposed and developed a model for Resource-Aware Ubiquitous Data Stream Querying (RA-UDSQ).

Our RA-UDSQ adapts the query process based on data rate, available memory and the rate at which the memory is being filled. Our RA-UDSQ model is iterative and time dependent and is focused towards continuous data stream queries.

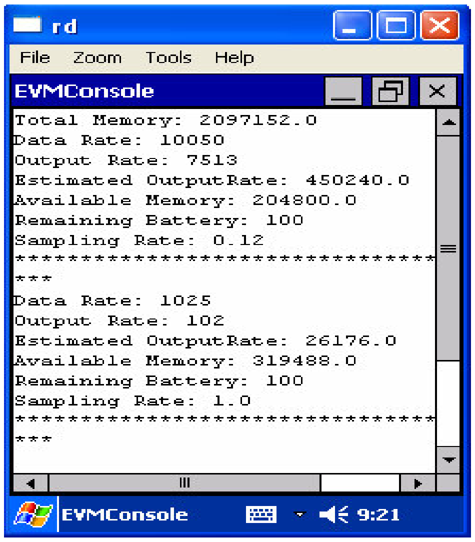
Our experimental evaluation of RA-UDSQ has shown the benefits of our approach as well as the impact of the adaptation process on memory savings, battery consumption and the approximation of results due to adaptation.

APPENDIX



```
1. Let D be the time duration.               // Phase 1 Identifying User
Constraints
2. Let TF be the current time frame.         // Phase 2 Optimization
3. Let N(TF) be the total number of time frames.
4. Let UTF be the unit time.
5. UTF = D / N(TF)
6. Let SR =1 be the initial sampling rate.
7. Repeat for every TF
     Calculate the data rate DR(TF).
     Calculate the starting time ST.
     Let n and j be the counting variables.
     Initialize output rate OR(TF) =0 and reading rate RR(TF)=0
     For each of the n iterations
          Perform sampling.
          If sampled data item
               Then
                    Increment RR(TF) by 1
                    Execute query on that record  // Phase 3 Executing
Query using
                    If record satisfies query          // Blocking
operators
                         Then
                              Increment OR(TF) by 1
     End For
     For each of the j iterations                      // Phase 4 Memory
Management
          Calculate estimated output rate EOR(TF) as
          EOR(TF) = [ [ Σ ( OR(TF)) / Σ (RR(TF) ) ] / TF ] * DR(TF)
     End For
     Calculate available memory M(TF)
     Compare EOR(TF) and M(TF)
     If EOR(TF) < M(TF)
        Then
          Let sampling rate SR =1
        Else
```

Error!
Figure 2 RA-UDSQ Algorithm



```
Total Memory: 2097152.0
Data Rate: 10050
Output Rate: 7513
Estimated OutputRate: 450240.0
Available Memory: 204800.0
Remaining Battery: 100
Sampling Rate: 0.12
********************************
***
Data Rate: 1025
Output Rate: 102
Estimated OutputRate: 26176.0
Available Memory: 319488.0
Remaining Battery: 100
Sampling Rate: 1.0
********************************
***
```

Figure 3 Values of data rate, output rate, available memory and remaining battery

Table 1 Memory Saving

Case No.	Time Duration in minutes	Total Available Memory in MB	RM(RA-UDSQ) in %	RM(UDSQ) in %	Difference in Available Memory (RM(RA-UDSQ) – RM(UDSQ)) at the end of time duration in %
1	10	4.98	5.82	4.22	1.60
2	15	4.01	14.46	10.72	3.74
3	20	5.44	3.68	3.13	0.55
4	25	4.40	5.23	4.32	0.91
5	30	4.00	6.75	3.00	3.75
6	45	6.84	7.89	3.22	4.67
7	60	4.83	10.77	5.80	4.97
8	90	5.27	3.80	3.61	0.19
9	120	9.28	4.31	1.19	3.12
10	180	4.58	4.80	4.15	0.65

Table 2 Case 5 RA-UDSQ Statistics

Case 5			
Time Frame	**Data Rates**	**Sampling Rates**	**Actual Data Items Read**
1	10,050	**0.34**	10,050
2	**6,225**	1	**2,137**
3	675	1	675
4	1,005	1	1,005
5	490	1	490

Table 3 Comparison of Final Output

Case No.	Total Data Rate (number of incoming data items)	Final Output (RA-UDSQ) in (%)	Final Output (UDSQ) in (%)	Difference of Final Output in (%)
1	11350	74.66	75.31	0.65
2	14789	74.72	75.59	0.87
3	27510	72.54	74.62	2.08
4	11320	70.48	75.28	4.80
5	18445	57.95	74.99	17.04
6	32720	67.58	74.97	7.39
7	25780	59.91	74.83	14.92
8	12645	74.36	74.38	0.02
9	33954	72.08	73.74	1.66
10	13515	72.48	75.21	2.73

REFERENCES

[1] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, J. Hwang, A. Maskey, A. Rasin, A. inger, M. Stonebraker, N. Tatbul, Y. Xing, R.Yan and S. Zdonik, Aurora: A Data Stream Management System (Demonstration)", Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'03), San Diego, CA, June 2003.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom, "Models and Issues in data stream systems" , Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS), Madison, Wisconsin, 2002, Pages: 1 – 16.

[3] M. N. Garofalakis, J. Gehrke, and R. Rastogi, "Querying and mining data streams: You only get one look (tutorial)" , Proceedings of ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, May 2002, Page: 635.

[4] M. M. Gaber, S. Krishnaswamy and A. Zaslavsky, "Adaptive Mining Techniques for Data Streams Using Algorithm Output Granularity" , The Australasian Data Mining Workshop (AusDM 2003), Held in conjunction with the 2003 Congress on Evolutionary Computation (CEC 2003) Canberra, Australia, Springer Verlag, Lecture Notes in Computer Science (LNCS), December 2003.

[5] M. M. Gaber, S. Krishnaswamy and A. Zaslavsky, "Ubiquitous Data Stream Mining" , Current Research and Future Directions Workshop Proceedings held in conjunction with The Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining, Sydney, Australia, May 26 2004.

[6] L. Golab and M. T. Ozsu, " Issues in Data Stream Management" , In SIGMOD Record, Volume 32, Number 2, June 2003, Pages: 5-14.

[7] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Resource-Aware Knowledge Discovery in Data Streams" , in the Proceedings of First International Workshop on Knowledge Discovery in Data Streams, in conjunction with ECML 2004 and PKDD 2004, Pisa, Italy, 20-24 September 2004.

[8] H. Kargupta, B. Park, S. Pittie, L. Liu, D. Kushraj and K. Sarkar, "MobiMine: Monitoring the Stock Market from a PDA" , ACM SIGKDD Explorations, Volume 3, Issue 2, ACM Press, January 2002, Pages: 37-46.

[9] S. Muthukrishnan, "Data streams: algorithms and applications" , Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms, 2003, Page: 413.

[10] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J.Rosenstein and R. Varma, "Query Processing, Approximation, and Resource Management in a Data Stream Management System" , Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR), January 2003, Pages: 245-256.

[11] S. D. Viglas and J. F. Naughton, "Rate-based query optimization for streaming information sources", Proceedings of the 2002 ACM SIGMOD International Conference on Management of data, Madison, Wisconsin, 03-06 June 2002, Pages: 37 – 48.