Earliest Deadline Scheduling for Continuous Queries over Data Streams

Xin Li¹, Zhiping Jia¹, Li Ma², Ruihua Zhang¹, Haiyang Wang¹

¹School of Computer Science and Technology
Shandong University
Jinan, China
e-mail: {lx,jzp}@sdu.edu.cn

²School of Computer Science and Communication Engineering
China University of Petroleum
Dongying, China
e-mail: mawawawa@gmail.com

Abstract—Many stream-based applications have real-time performance requirements for continuous queries over time-varying data streams. In order to address this challenge, a real-time continuous query model is presented to process multiple queries with timing constraints. In this model, the execution of one tuple passing through an operator path is modeled as a real-time task instance. A fine-grained scheduling strategy named OP-EDF is proposed for real-time scheduling, which schedules the operator path with the earliest deadline of the waiting tuples at any time slot. The performance of the OP-EDF is analyzed from three aspects: schedulability, response time and system overhead. Furthermore, two improved batch scheduling algorithms, termed OP-EDF-Batch and OP-EDF-Gate, are introduced to decrease system overhead of the OP-EDF. The experiment results show that the proposed continuous query model and improved scheduling algorithms are effective in real-time query processing for data streams with bursty arrival rates.

Keywords—data stream; continuous query; real-time; scheduling; deadline

I. INTRODUCTION

A large number of information processing applications, such as traffic engineering, stock trading and network monitoring, have to manipulate high volume stream data in a timely manner [1, 2]. These applications have sparked researchers’ interest in the area of data stream management system (DSMS) in both the database and real-time computing communities. DSMS is designed to process a large number of concurrent continuous queries over rapid, time-varying, possibly unpredictable and unbounded streams. A growing body of research rage from synopsis and algorithms for stream processing to prototype systems such as Aurora [6], STREAM[11], TelegraphCQ [8]. Run-time resource allocation and optimization is one of the most important components in DSMS. In this paper, we focus on CPU time allocation and load management in real-time circumstance.

In data stream model, individual data items may be relational tuples, e.g., sensor readings. Streaming applications are used to monitor the status of the objects (e.g., pipe pressure) in the external environment and they must generate timely responses to critical events. In these systems, queries have to be completed by certain deadlines for the results to be full of value. These requirements constitute a distinct class of queries that perform real-time processing of data describing a physical world into information useful to end applications. This raises the problem of allocating system resources (e.g., CPU and Memory) to query operators, i.e., how to schedule the execution of the operators for high performance.

Aimed at the above problems, we introduce deadlines to characterize the timing constraints of continuous queries. A continuous query with deadline is regard as a real-time task and one triggered execution of a continuous query as one task instance. And we build up a real-time task model for continuous queries. Then an Earliest Deadline First based on Operator Path (OP-EDF) algorithm is proposed. Its objective is to minimize deadline Miss Ratio (MR), i.e. the percentage of query instances that do not complete before their deadlines. Furthermore, two improved scheduling algorithms are proposed to reduce scheduling overhead by batching tuples. The experimental results show that the proposed continuous query model and improved scheduling algorithms are effective in real-time query processing for data streams with bursty arrival rates.

The remainder of this paper is organized as follows: Section 2 gives the related work. Section 3 describes the system overview and real-time query model. An EDF-based query scheduling is proposed and analyzed in section 4. And then, section 5 presents two improved scheduling strategies to reduce the switching times. Section 6 gives the performance evaluation and experimental results. Finally, section 7 summarizes our conclusions and identifies future work.

II. RELATED WORK

Streaming applications have sparked researchers’ interest in the area of DSMS in both database and real-time computing communities. They have proposed several scheduling algorithms over data streams for better system performance. The Chain scheduling strategy [11] aims to minimize the maximal memory usage. The Aurora project employs a two-level scheduling approach [8], which has three variations targeting respectively to minimize average tuple la-
tency, maximize query throughput or minimize total queue size. The Rate-based scheduling [13], aimed at maximizing the throughput of DSMS, assigns higher priority to the operator which has higher output rate. The PC Strategy [10] achieves the overall minimal tuple latency. However, these strategies including PCS, Chain and Rate-based schedule continuous queries without timing constraints and they are not real-time scheduling.

There has been some work that discussed the real-time query processing and scheduling over data stream. Wei et al. [4] uses EDF strategy to schedule the periodical queries. Schmidt et al. [6] discusses a hard real-time scheduling strategy based on Rate Monotonic (RM) and assumes that fluctuation of stream rates is small and all operators have running period. The QoS-driven scheduling [7,8] in Aurora project claims to provide soft real-time processing capabilities and their real-time performance metric is the average latency of data tuples not deadlines. ATS[14] and Tick scheduling[17] strategy is proposed to maximize task throughput and minimize deadline miss ratio by minimizing both scheduling overheads and deadline miss overheads. We proposed an adaptive load management strategy [15] based on dynamic execution time prediction to maximize to overall query quality. In this paper, we focus on the scheduling of continuous query with timing constraints and improve QoS of continuous queries in real-time circumstance.

III. REAL-TIME QUERY MODEL

In this section, an overview of the real-time DSMS is given firstly, and then a real-time query model is presented for time constraints.

A. Data Model

A data stream is defined as a real-time, continuous, ordered (implicitly by arrival time) sequence of data items [1]. It is impossible to control the arrival rates and contents of the data streams. Due to the high volume of data streams, it is impossible to store all data in memory or on disk in its entirety and it is unfeasible to access a whole stream history. Typically, queries are constrained to process data inside a sliding window, which is a recent segment of data stream. Although the memory constraint is a very important and complicated research problem, in this paper, we only consider the slow-CPU problem and assume that there is always sufficient memory for query processing.

Depending on the extent to which applications can tolerate violations of their time constraints, a real-time system can be characterized as being either hard, soft or firm [3]. Our target is to provide firm real-time query services, in which overdue results have no value and may not yield a catastrophic disaster for the applications. The conception of query deadline in data stream system is defined as follows.

Definition 1. (Query Deadline) Assume that Q is a continuous query over data streams S = {s1, t1}, <s2, t2>, ..., <sn, tn}, in which si denotes a data tuple in S and ti denotes the arrival time at which si enters into DSMS. The relative deadline D denotes the time by which si should be processed completely in query Q, i.e. the tuple becomes qualified results for the application or fails to meet the query conditions. The query Q with a relative deadline is called a real-time query and (t + D) is called the absolute deadline of a tuple si in query Q.

The relative deadline is specified by applications when they register a query into the DSMS. A continuous query example written in RT-CQL[5] is given as follows.

Query 1. On every data arrival, monitoring the positions of those vehicles on express way #12 whose speeds are more than 60 mph. with deadline 10 seconds.

SELECT Vehicle_ID, x_pos, express_way
FROM PositionStream
WHERE speed>60 and XWay=12
DEADLINE 10 seconds

Query results are useful for the applications only if they are produced within the deadline (10 seconds).

Definition 2. (Deadline Miss Ratio) Deadline Miss Ratio(MR) denotes the percentage of query instances that do not complete before their deadlines.

\[ MR = \frac{N_i^M}{N_i} \]  \hspace{1cm} (1)

Where Ni and Ni^M are the count of all instances of the i-th query and the one of those miss their deadlines respectively. If there are k queries, the system miss ration is defined as:

\[ MR_{sys} = \frac{\sum_{i=1}^{k} N_i^M}{\sum_{i=1}^{k} N_i} \]  \hspace{1cm} (2)

The MR_{sys} is lower, the QoS of the system is better.

B. System architecture

Figure 1 illustrates the system architecture of the real-time DSMS called RT-DSMS [14] extended from STREAM [12] prototype. The system consists of five components: Data Source Manager, Query Processing Engine, Real-Time Scheduler, QoS Monitor and Load Shedder.

Data Source Manager receives data from multiple streams and inserts data into corresponding input queues of query plans as well as monitors input characteristics of streams (e.g. streams rates). The Query Processing Engine compiles user-defined queries into query plans and optimizes them dynamically. We adopts query processing
mechanism [18] based on pipelines in which query is translated to query execution plan composed of basic operators and data queues. For example, query 1 is translated to a plan of one projection operator and two selection operators (one for speed>60, the other for XWay=12).

Multiple query execution plans are combined into an operator execution graph, also called operator network in which a node (yellow circle in Fig.2b) represents an operator and an edge represents a data queue between two consecutive operators. A tuple have to pass through a unique path of operators, i.e. Operator Path (OP). In other words, operators in an OP process one tuple in sequence.

Assume that there is enough memory space for DSMS, while CPU is not fast enough for query processing when arrival data rates are bursty.

C. Real-time task model

A continuous query with deadline is defined as a real-time task and one execution (triggered by a tuple arrival) of a continuous query as one task instance[16]. Every query \( Q \) consists of a sequence of instances \( q_i(l), \ i \geq 1 \). The \( j \)-th instance \( q_j(j) \) can be depicted as \( \langle a_j(j), d_j(j) \rangle \), in which
- \( a_j(j) \) denotes the arrival time when the tuple enters.
- \( d_j(j) \) denotes the absolute deadline of the instance.

\[ d_j(j) = a_j(j) + D_i \]

where \( D_i \) is relative deadline of \( Q_i \).

If multi-queries share the same path, \( D_i \) equals minimum deadline among these queries. If there are multiple tuples queuing for processing, The \( D_i \) is equal to \( \min_{\forall \text{query}} \{ a(x) \} + D_i \). \( Q \) consists of a sequence of instances \( q_i(l), \ i \geq 1 \). The \( j \)-th instance \( q_j(j) \) can be depicted as \( \langle a_j(j), d_j(j) \rangle \), in which
- \( a_j(j) \) denotes the arrival time when the tuple enters.
- \( d_j(j) \) denotes the absolute deadline of the instance.

\[ d_j(j) = a_j(j) + D_i \]

where \( D_i \) is relative deadline of \( Q_i \).

Assume that the \( i \)-th operator path consists of \( n_i \) operators, the \( j \)-th operator’s probabilistic selectivity is \( s_{ij} \), and the cost time of processing a tuple is \( c_{ij} \), then estimation execution time of a task instance is

\[ e_i = \sum_{j=1}^{n_i} c_{ij} \prod_{k=1}^{j-1} s_{ik} \]

\[(3)\]

A set of operator paths with shared operators would constitute an OP Cluster. In this cluster, path with minimum relative deadline is called as Key Path, and other paths are called as Branch Paths. Figure2(c) shows a cluster composed of three paths. \( a \rightarrow b \rightarrow c \) is a key path, and \( d \rightarrow e \) and \( f \rightarrow g \rightarrow h \) are branch paths. When tuples arrive in cluster, system schedules firstly operators on key path, then branch paths ascended by deadlines.

There are hundreds of queries usually in a DSMS. If every query is assigned a thread, the context-switch cost and synchronous cost of multiple threads will be very high. In RT-DSMS, real-time scheduler is realized as one high priority work thread to avoid the switching overhead of multiple threads. The scheduling strategy in scheduler decides the execution order of tasks, and endeavors to complete each task within its deadline.

Assume that the number of operator paths is \( m \), the number of parts of shared paths is \( l \), and input stream rate of the \( i \)-th path is \( \lambda_i \), then utilization ratio of the system \( U_{SYS} \) is computed as follows:

\[ U_{SYS} = U_{ALL} - U_{SHARE} = \sum_{j=1}^{m} (\lambda_j \times e_j) - \sum_{j=1}^{l} \lambda_j \times (f_j - 1) \times H_j \]

\[(4)\]

Where \( f_j \) and \( H_j \) respectively the number of shared paths (total sharing and part sharing) and estimation execution time \( (H_j, e_j \) are computed from (3)). Assume that the maximum CPU utilization ratio is \( \delta \) (0 ≤ \( \delta \) ≤ 1). When \( U_{SYS} \) is less than \( \delta \), the system would complete query processing of all tasks.

The merits of this model are following: (1) Tasks can be executed independently, and there is no dependency be-
between tasks; (2)scheduling strategy can be preemptive (new task with earlier deadline preempt the current task instance and an operator processing a tuple is primitive), and also be non-preemptive.

IV. EDF-BASED QUERY SCHEDULING

We firstly discuss the data structure of scheduling algorithm, and then present the OP-EDF algorithm.

A. Data structure

To schedule efficiently, an Operator Path Table (OP Table) and an active queue are created based on operator execution graph. As shown in Figure 2, OP Table is an array of OP nodes. It saves the state of each path and the pointer pointed the first operator. Active queue is a double linked list, and only links OP nodes those input or internal queues are not empty. Active queue is ordered by absolute deadlines of the earliest tuple in paths.

B. OP-EDF strategy

OP-EDF is a two level scheduling algorithm with dynamic priority. (1)It always executes the OP with the earliest absolute deadline. If multiple tasks have the same deadline, it selects the earliest arrived task. (2)Within an OP, operators are scheduled from beginning to end. After a tuple is processed by an OP, the scheduler checks input queue, deletes overdue tuples, and updates the deadline of this OP as the deadline of the earliest tuple in the OP. The OP-EDF algorithm is shown as Figure 3.

Algorithm: OP_EDF_Scheduler()

Input: Active Queue: AQ
Output: the operators sequence to execute
1. CurrOP=GetFirstOperatorPath(AQ);
2. OP_Internal_Scheduler(CurrOP); //call internal strategy
3. NextOP= CurrOP->NextActiveOP
4. RefreshActiveQueue(); //update deadlines of OPs
5. WHILE (NextOP<>NULL) DO
6. IF (NextOP->Deadline>GetCurrentTime()) THEN
7.   BREAK;
8. ELSE
9.   NextOP->HandleOverdue(); //delete overdue tuples
10. END IF
11. END WHILE

Figure 3. OP-EDF Scheduling Algorithm

OP_Internal_Scheduler(CurrOP) is a preemptive scheduling algorithm based on tuples, and adopts FIFO strategy to process tuples. When the algorithm is executed, it firstly checks whether there exist preempted tuples. If so, it processes these tuples, or else selects the earliest arrived tuple among input queue to process. Because of limited space, this paper omits the detailed description of OP_Internal_Scheduler(CurrOP).

C. Performance Analysis

1) Schedulability

OP-EDF is an optimal uniprocessor scheduling algorithm. i.e., if a set of queries can’t be feasibly scheduled under OP-EDF; there is no other uniprocessor algorithms that can schedule them successfully.

Theorem 1. OP-EDF is optimal for uniprocessor.

Proof. (By contradiction)

Suppose that the theorem is not true and there is another algorithm Ψ which is optimal. It’s certain that there exist a set of query instants S that is Ψ-schedulable but not OP-EDF schedulable.

Suppose that t₂ is the earliest absolute deadline that is missed by the OP-EDF algorithm. Define t₁ as the last instant, prior to t₂, when OP-EDF had the processor working on a task whose absolute deadline exceeded t₂. If no such instant exists, set t₁=0. Since only tasks with absolute deadlines ≥ t₂ are scheduled by OP-EDF in the interval [t₁,t₂], any instance executing in that interval must have been released at or after t₁. The reason is that at t₁, the processor was executing a task with an absolute deadline > t₂, which would only have been possible under OP-EDF if there was no suspended task at t₁ with an absolute deadline ≤ t₂. Define

A={q| q is released in [t₁,t₂] and D< t₂}
B={q| q is released in [t₁,t₂] and D≥ t₂}

According to the definition of t₂, B is non-empty, moreover all the deadlines of the tasks in A are met by both OP-EDF and Ψ. There are two cases.

Case 1. Under OP-EDF, the processor is continuously busy over (t₁,t₂).

Let T^OP-EDF(A), T^OP-EDF(B), T^Ψ(A), T^Ψ(B) be the execution time over (t₁,t₂) allocated by OP-EDF and Ψ to the tasks in A and B, respectively. Then,

T^OP-EDF(A)+T^OP-EDF(B)=t₂−t₁ (5)

Since all the deadlines of tasks in A are met by both OP-EDF and Ψ,

T^OP-EDF(A)=T^Ψ(A) (6)

However, since at least one instance in B misses its deadline under OP-EDF, we must have

T^OP-EDF(B)<T^Ψ(B) (7)

Hence, in the interval (t₁,t₂], under Ψ, the processor is used for

T^Ψ(A)+T^Ψ(B)>T^OP-EDF(A)+T^OP-EDF(B) (8)
T^Ψ(A)+T^Ψ(B)>t₂−t₁ [from (5) and (8)] (9)

But that is plainly impossible, and a contradiction is deduced.

Case 2. Under OP-EDF, the processor is idle over some part of (t₁,t₂).

Let t₃ be the last instant in (t₁,t₂) at which the processor is idle under the OP-EDF strategy. Since OP-EDF causes a deadline to be missed at t₃(t₃<t₂).
The processor can only be idle at \( t_1 \) if there are no suspended instances for execution, that is, if every instance released prior to \( t_1 \) has been executed. The argument we made in Case 1 over the interval \( (t_1,t_2] \) now applies over the interval \( (t_2,t_3] \), and a contradiction is deduced.

To sum up, the assumption is error. i.e., OP-EDF is optimal for uni-processor.

\[ \blacksquare \]

2) **Response time**

Chain [11], PCS [10], Rate-base [13] and RM [6] schedule operators with static priority. In these strategies, low-priority operators or OPs have no chances to be scheduled if high-priority ones are always triggered. This causes low-priority queries or operators to delay unboundedly, and these queries will experience starvation. OP-EDF is a dynamic-priority strategy and can avoid the starvation problem.

**Theorem 2.** OP-EDF strategy can avoid the starvation problem.

**Proof.**

OP-EDF always schedules query instances according to their absolute deadlines and executes the one with the earliest deadline. Therefore, no matter what the relative deadline and query cost are, the instance must be scheduled if there is idle time for CPU before its absolute deadline. The high-cost OPs will not be preempted by an unbounded number of instances. Therefore, OP-EDF can avoid starvation problem.

In addition, since OP-EDF schedules tuples one by one, the results are smooth and prone to be used by query applications.

\[ \blacksquare \]

3) **Scheduling overhead**

On the basis of the OP-EDF strategy, the scheduler chooses an instance with the earliest deadline in all waiting OPs every time. Assume that there are \( n \) OPs to be scheduled, the complexity of sorting all OPs by their deadlines is \( O(n \log_2 n) \) at least. We decrease the cost by maintaining a sorted ready queue (Figure 2b). This is an incremental sort process including two steps: (1) compute the absolute deadline of an instance and insert the instance into an appropriate position according to its deadline. The complexity of inserting an element into a sorted queue is \( O(n) \); (2) select the first OP to execute from active queue. The complexity is only \( O(1) \). Therefore, the scheduling overhead of OP-EDF is \( O(n) \).

The cost of QoS-driven scheduling [8] and LSF (Least Slack First, in which task with the smallest slack time is scheduled first) is higher than OP-EDF because the priorities of all tasks in these two strategies have to be recomputed on every scheduling switching.

Since the OP-EDF processes one tuple every time, the frequency of the context switching is linear to the sum of streams rates. The faster streams arrive, the more switching times are. In the worst case, when streams burst, context switching is frequent and system performance gets worse sharply.

**V. STRATEGY IMPROVEMENT**

To reduce the switching times, we propose two improved internal schedule strategies: OP-EDF-Batch and OP-EDF-Gate. They process multiple tuples as a batch within an operator running in order to cut down system overhead.

**A. OP-EDF-Batch strategy**

OP-EDF-Batch executes a sequence of tuples as a batch task. After the processing of a batch task finishes, the scheduling switches again. The deadline of a batch task is equal to the deadline of the earliest tuple in the batch. The OP-EDF-Batch implements batch processing by assigning each operator when to execute and how many queued tuples to process. The number of tuples in one batch is decided by two parameters: Time slice \( \delta_T \) and tuples threshold \( \delta_b \). When the longest waiting time is larger than \( \delta_T \), or the number of tuples in a queue is larger than \( \delta_b \), the processor is idle, the scheduler sets the path first to process.

Time slice \( \delta_T = \frac{\text{GCD}(D_1, D_2, \ldots, D_n)}{T} \), where \( \text{GCD}(D_1, D_2, \ldots, D_n) \) denotes the greatest common divisor of all queries’ relative deadlines, and \( T \) is time-slice parameter and its value is integer greater than 1. \( \delta_b = \text{MAX}(\lambda_1, \lambda_2, \ldots, \lambda_m) \), where \( \text{MAX}(\lambda_1, \lambda_2, \ldots, \lambda_m) \) denotes the maximum of all streams rates. Amounts of data may arrive in the same time slice in case of data bursty. To prevent that the processing time of this batch tasks is too long and affect fair execution of other queries, OP-EDF-Batch limits the number of tuples in each batch task by \( \delta_b \). Because \( \beta \) affects \( \delta_T \) and \( \delta_b \), the size of \( \beta \) greatly affects scheduling switch and performance of the strategy. With the increasing of \( \beta \), the granularity of scheduling decreases. At the same time, the flexibility of scheduling, the frequency of switch and overhead increase. If \( \beta \) is too large, OP-EDF-Batch algorithm degenerates into OP-EDF algorithm. If \( \beta \) is too small, the real-time of scheduling and accuracy can not be guaranteed. So the optimal value of \( \beta \) is determined through experiments in this paper.

Given a fixed number of tuples to process, OP-EDF-Batch decreases the total number of operator executions required to process those tuples, thereby cutting down scheduling overhead.

**B. OP-EDF-Gate strategy**

OP-EDF-Gate is also a batching internal schedule strategy similar to “bottom-up”. It executes operators along the operator path, and processes all tuples in an operator’s input queue. Then it schedules next operator with all tuples. This strategy greatly reduces the number of operator scheduling, and then reduces system overhead. Like OP-EDF-Batch, OP-EDF-Gate also can greatly improve the system performance in the case that streams rates burst.
VI. EXPERIMENTS

We have developed a real-time DSMS, termed RT-DSMS[5], to evaluate our query model and scheduling strategies. Several experiments are made in the system using simulated data and real-world data sets. All experiments are carried out on a PC machine with a Pentium 4 processor (2.4 GHz) and 2GB main memory.

A. Experiment setting

At first, we simulate 8 streams with the same format (ID, Temp, Humid) and 16 continuous queries. The detail simulation parameters are given in Table 1. The selectivity and cost of operator are fixed for simplicity. Data streams arrive in accordance to a Poisson distribution. The mean arrival rates range from [100, 5000]. The values of Temp and Humid are simulated in uniform distributions.

Three experiments are presented to test the performance of OP-Round-Robin (OP-RR), OP-LSF, OP-EDF, OP-EDF-Batch (OP-EDF-B) and OP-EDF-Gate (OP-EDF-G), and the effect of time-slice parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Size</td>
<td>4K</td>
</tr>
<tr>
<td>Operator Cost</td>
<td>15-80μs</td>
</tr>
<tr>
<td>Operator Selectivity</td>
<td>0.1-1.0</td>
</tr>
<tr>
<td>Stream Number</td>
<td>8</td>
</tr>
<tr>
<td>Query Number</td>
<td>16</td>
</tr>
<tr>
<td>Query Depth</td>
<td>1-4</td>
</tr>
<tr>
<td>Query Fan-out</td>
<td>1-2</td>
</tr>
<tr>
<td>Query Deadline</td>
<td>10-100ms</td>
</tr>
</tbody>
</table>

B. Effect of different stream rates

In this experiment, we test the performance of different algorithms under different streams rates. Here, the OP-EDF-Batch time-slice parameter \( \beta = 10 \). During 10 minutes of each experiment time, the deadline Miss Ratio (MR) and the system overhead are recorded.

Figure 4 shows the system miss ratios under different algorithms. It indicates that the miss ratio of OP-RR algorithm is higher than other algorithms because it does not take query deadlines into consideration and thus it is not suitable for real-time scheduling. The OP-LSF and OP-EDF strategies, which are both dynamic priority scheduling strategies and only slightly different on methods of calculating task priority, are close to each other on performance. When streams rates are below 3000 tuples/sec and query load is under system’s maximum capability, the miss ratio in OP-LSF, OP-EDF and improved algorithms are all low. However, as stream rate and query load increase, the miss ratio increases continuously. When stream rate surpass 3500 tuples/sec, the miss ratio of OP-EDF algorithm increases linearly and is extraordinarily higher than the ones of the improved algorithms. On the other hand, OP-EDF-Batch and OP-EDF-Gate algorithms can still keep low miss ratio (smaller than 0.2) under high stream rate. In addition, compared with OP-EDF-Gate algorithm, the miss ratio of OP-EDF-Batch algorithm is lower. This is because scheduling granularity of OP-EDF-Batch is lower than that of OP-EDF-Gate and average tuple output latency is relatively shorter. We can see that adaptabilities of OP-EDF-Batch and OP-EDF-Gate algorithms to variable stream rate are stronger than that of OP-EDF and OP-LSF algorithms and thus are more suitable for instances when stream rate varies markedly.

Figure 5 shows system overhead under different algorithms. OP-LSF scheduling has to compute the earliest waiting time and priority of tuple in all operator paths, while OP-EDF only needs to compute priorities of those tasks that changed; thus system overhead of OP-EDF is lower than OP-LSF. The system overhead of improved algorithms is significantly lower than that of OP-EDF and OP-LSF algorithms. Along with the increasing of stream rate, system overhead increases gradually. When stream rate surpasses 3250 tuples/sec, system overhead of OP-EDF and OP-LSF increases quickly while the one of OP-EDF-Batch and OP-EDF-Gate increases slightly. The reason is that OP-EDF algorithm processes only one tuple during each scheduling switch. When system is overload, switching overhead increases markedly. On the other hand, as the stream rates accelerate, the scheduling switch frequencies in OP-EDF-Batch and OP-EDF-Gate increase a little and system overhead is low relatively. The switch frequency of OP-EDF-Gate scheduling is lower than that of
OP-EDF-Batch, and thus system overhead of OP-EDF-Gate is much lower. These results match theoretical analysis in section 5.

C. Relative execution overhead

In order to compare execution overhead of these algorithms, we choose a real workload, position tuples of the first-30-minute data from Linear Road Benchmark [9] (about 500,000 records, maximum stream rate equals 610 tuples/sec), which have the following format:

Pos (Type=0, Time, VID, Speed, XWay, Lane, Dir, Seg, Pos).

In this experiment, there are four queries and six operators in each query. The stream simulator loads all data file into memory firstly, and then sends tuples to data source manager according to their timestamps. The QoS Monitor collects the execution time of each algorithm. Execution time is primarily consisted of system overhead, including scheduling cost, and operator cost. Operator cost is equal to the processing time of all operators.

Figure 6 shows that the total execution time of OP-EDF algorithm for processing the data is 35.2 seconds, which is entirely higher than that of OP-EDF-Batch (15.3 sec) and OP-EDF-Gate (11.6 sec). In OP-EDF algorithm, system overhead holds a high ratio and is about 12.5% of the overall execution time, while in OP-EDF-Batch and OP-EDF-Gate they are respectively 4.3% and 3.6%. Hence, it can be seen that improved algorithms reduce system cost significantly and improve processing efficiency. In addition, because operators can process more tuples in one batch in OP-EDF-Gate algorithm than in OP-EDF-Batch, scheduling switch frequency and system overhead of OP-EDF-Gate is lower. That is to say, valid processing time occupies a higher ratio in OP-EDF-Gate than that in others, and OP-EDF-Gate gets higher execution efficiency.

D. Effect of different time-slice parameters

The performance of OP-EDF-Batch algorithm is affected by time-slice parameter $\beta$, especially when stream rates burst. The third experiment tests the influence of different $\beta$ to the algorithm using simulation data (simulation parameters refer to Table 1). In this experiment, the rates of eight streams are all equal to 4000 tuples/sec; stream burst[8] are setting as follows: (1) $\text{burst}=1$, data arrive in according to uniform distribution in one second; (2) $\text{burst}=5$, data arrive at time $t=\text{i}/800(i=0,1,2,...,799)$ with 5 tuples each time; (3) $\text{burst}=10$, data arrive at time $t=\text{i}/400(i=0,1,2,...,399)$ with 10 tuples each time.

Figure 7 shows that, in the three settings, the miss ratio varies between 0.12–0.17, and the fluctuating range is not wide. This result indicates that this algorithm is not easily influenced by stream bursts and has a strong adaptability to stream rate variation. The miss ratio of OP-EDF-Batch algorithm is greater in higher stream burst than the one in lower stream burst. In each setting ($\text{burst}=1, 5$ or $10$), the miss ratio declines first and then increases. This is especially evident when $\text{burst}$ equals 10. With the increasing of $\beta$ (from 2 to 10), the miss ratio declines from 0.157 to 0.143. After $\beta$ surpasses 12, the miss ratio climbs evidently. We can conclude that there lies a balance point between scheduling cost and switching in time. Only when time-slice $\beta$ is set at proper range (the optimum $\beta$ ranges from 8 to 12 in this experiment), can OP-EDF-Batch algorithm show the best performance.

VII. Conclusion and future work

This paper introduces a fine-grained real-time scheduling strategy named OP-EDF, in which operator path with earliest deadline is scheduled firstly. In order to decrease the system overhead, two improved algorithms are given and they achieve higher processing efficiency by batching method.

In this paper, we only consider one objective (timing constraints) and neglect the memory constraints. For future work, we plan to design an adaptive real-time scheduling strategy involving the requirements of CPU and memory usage. Another work is to explore ways to load shedding and admission control in the real-time query model when the system is overloaded.

Acknowledgment

This research is sponsored by the Natural Science Foundation of China (NSFC) under Grant No. 90718032 and by the Youth Faculty Award of the School of Computer Science and Technology of Shandong University in China.
REFERENCES


