Adaptive Run-time Overhead Adjustments for Optimizing Multiple Continuous Query Processing*

Hyun-Hon Lee¹, Hong-Kyu Park², Jin-Chul Park³, Won-Suk Lee⁴, Kil-Hong Joo⁵†

¹Dept. of Computer Science, Yonseong University, Korea
²Samsung Electronics Co. Ltd., Korea
³Dept. of Computer Science, Hanyang University, Korea
⁴Dept. of Computer Science, Yonsei University, Korea
⁵Dept. of Computer Education, Gyeongin National University of Education, Korea

¹hhlee@yeonsung.ac.kr, ²gladiatorl1@hanmail.net, ³jin1chul@nate.com, ⁴leewo2001@gmail.com, ⁵khjoo@ginue.ac.kr

Abstract

The time-varying characteristics of infinite data streams require continuous queries to be adaptively processed. The order in which multiple join operations are evaluated has serious consequences for the algorithm performance because the selectivity of each join operation can differ significantly from the selectivity of the other operations. The evaluation order may be effectively determined using the k-EGA and A-SEG0 schemes, as proposed in previous studies. These methods optimize target continuous queries by monitoring a set of their promising subplans simultaneously. Each scheme also employs a user-defined cost-bound parameter for controlling the number of monitored subplans. A more optimized global plan may be generated by using a more highly configured cost-bound parameter. However, this approach can increase the overhead associated with monitoring the subplans. This paper proposes a new scheme, Adaptive Run-time Overhead Adjustment (AROA), which provides a novel method for adaptively determining the value of a cost-bound parameter based on the system environment. Unlike the previously described A-SEG0 scheme, the scheme proposed here automatically selects the cost-bound parameter to reflect the system workloads (e.g., the input tuple rate, and other parameters). This method not only augments the probability of generating an optimized execution plan, it reduces the run-time delay caused by the optimization process. Experimental verification of the proposed scheme AROA demonstrated that AROA outperforms the previous schemes.

Keywords: data stream, continuous query, join operation, AROA, adaptive optimization, cost-bound parameter

1. Introduction

Data streams, which are a representative type of big data, are massive unbounded sequences of data elements that are continuously generated at a rapid rate. Data streams are used in many emerging applications, such as web click monitoring, sensor data processing, network traffic analysis, telephone record analysis, and multimedia. Ongoing changes in the rapid data streams in a data stream management system

* This paper is a revised and expanded version of a paper entitled “A New Continuous Query Evaluation Scheme for Multiple Data Streams” presented at NGCTT2014.

† Corresponding Author
(DSMS) may be monitored using a number of pre-registered queries that report results continuously as new data elements of the data streams arrive [1, 2, 3]. Such queries are called continuous queries. Due to the strict time constraints of the continuous queries in a stream environment, query optimization processes are essential.

The optimization techniques used for continuous queries are quite different from those used in traditional queries in the following respects. First, a continuous query requires memory-resident data for blocking operations, such as grouped filters and windowed joins [1, 7-10]. Second, a query’s execution plan should be re-optimized adaptively due to the time-varying characteristics of the data streams [11]. In other words, whenever the current plan is no longer optimal, a new optimized plan should be generated. The overhead required to generate a new plan should be minimized to ensure an effective plan change. Previous studies have proposed a multi-way join algorithm, k-EGA, for the adaptive production of an optimized execution plan for a single continuous multi-way join over a data stream [12]. A multi-way join consists of multiple binary joins. k-EGA concurrently monitors promising sub-execution plans corresponding to the binary joins. The execution costs associated with each subplan are estimated by maintaining a cost profile containing the statistics on the actual costs of the corresponding join operation. The optimized execution plans of a multi-way join are then determined based on the cost profiles by extending the evaluation sequence of binary joins in a greedy manner. The k-EGA algorithm has been modified to give a multiple query optimization method, A-SEGO [13]. This algorithm provides an optimized global execution plan for multiple continuous multi-way joins by sharing the on-going results of their common binary join operations. As with the k-EGA algorithm, the A-SEGO method determines the complete global plan for the multiple multi-way joins in a greedy manner based on the actual costs of the binary join operations consisting of multi-way joins; however, unlike the k-EGA algorithm, the A-SEGO algorithm additionally considers the degree of sharing among the binary join operations. The degree of sharing is measured according to the number of specified binary join operations that may be reused.

Both proposed methods, k-EGA and A-SEGO, adaptively change the execution plans of target continuous queries based on their own cost profiles. In the cost profiles, a user-defined parameter called a cost-bound parameter is employed to control the number of concurrently monitored candidate subplans. According to its definition, as its value is increased, the probability of finding an optimal global plan is increased but the overhead of monitoring the candidate subplans is also increased. This overhead can be measured by the time that it takes to generate the new optimized global plan. It is called an optimization duration. In order to optimize the overall performance of query execution including the monitoring overhead, the optimized plan should be generated within the optimization duration that the current system environments can allow. Because the optimization duration is very closely associated with the value of cost-bound parameter, it is important to find the allowable optimization duration in order to set up the cost-bound parameter. This paper proposes an adaptive optimization scheme called an Adaptive Run-time Overhead Adjustment (AROA) based on the A-SEGO scheme. Instead of defining the cost-bound parameter manually as it did in the previous studies, the proposed scheme adjusts the parameter automatically, considering the current system workload. For this purpose, the proposed scheme takes advantage of the past statistics that experientially specifies the relationship between a cost-bound parameter and an optimization duration. It calculates the maximum optimization duration under the current system environments. Subsequently, it determines the highest
cost-bound parameter that is allowable for the acquired optimization duration, referring to the past statistics. It makes a set of target multi-way join queries to be evaluated with the most efficient execution plan under the current system environments.

The remainder of this paper is organized as follows: Section 2 presents related work. In Section 3, the A-SEGO scheme is described as the preliminaries. Section 4 proposes how to determine the value of the cost-bound parameter automatically. In Section 5, the performance of the proposed method is analyzed through a series of experiments. Section 6 presents our conclusions.

2. Related Works

Multiple continuous queries may be evaluated effectively in DSMS using a variety of proposed optimization methods. STREAM [14] proposed the A-Greedy algorithm to adaptively determine the optimal evaluation order for certain pipelined filters. This algorithm may be extended to determine the order in which the join operations of a single query are executed in an MJoin operation [15]. More precisely, multiple binary join operations may be collapsed into a single MJoin operation without the need for a tree-structured execution plan. The execution result is then generated without materializing the intermediate results of any binary join operations. For an incoming tuple of a source data stream, the evaluation order of the join operations can be reordered dynamically to minimize the overall evaluation cost of the operations. The on-going selectivity change of the evaluation order is continuously monitored to adaptively adjust it. Accordingly, the overall processing cost can be minimized at all times even if the join selectivities are varied unpredictably over time [7, 16]. Eddy [2] achieves the fine-grained adaptivity for multiple query optimization without employing any fixed execution plan. This query execution model employs the ticket-based routing policy that sends input and intermediate tuples to the operations of multiple queries. By deciding the routing path of each tuple independently, the dynamical change of the selectivities of the operations can be reflected on their execution order. CACQ [1] is based on the Eddy framework. It supports not only a grouped-filter index to evaluate multiple selection operations together but also a space-efficient double-pipelined hash join method called SteMs [17]. In [18], as a modification for the original Eddy framework that uses SteMs, a STAIR (Storage, Transformation and Access for Intermediate Results) is introduced. It employs a BLPT (Binary Linear Processing Tree) [4] as a fixed execution plan. For a continuous query, based on the selectivities of all the subplans for its join predicates, a greedy optimization approach is employed to determine the overall execution plan of STAIRs dynamically. A JTree-Finder algorithm [19] generates a qualified execution plan, which is a tree of MJoin operations. It guarantees that both of CPU and memory usages for evaluating the execution plan stay within the system capacity. A OptDP algorithm [20] employs dynamic programming to find an optimized execution plan for a single multi-way join query over data streams. However, it does not consider sharing the common sub-expressions of multiple queries. As our previous studies, k-EGA algorithm [12] produces an optimized execution plan of a single multi-way join query over data streams by concurrently tracing a set of promising subplans. Also, A-SEGO algorithm [13] produces an optimized global execution plan for the collective evaluation of multiple continuous queries by sharing the ongoing results for the common join operations of a set of multi-way join queries.
3. Preliminaries

Given a set of multi-way join queries \( Q = \{ q_1, q_2, \ldots, q_n \} \), A-SEGO generates an optimized global execution plan by examining the subplans consisting of the global execution plan of the query set \( Q \) without the use of a backtracking mechanism. The method decreases the processing costs associated with the query set \( Q \) by sharing the results of common sub-expressions for its multi-way join operations. The overall A-SEGO process is as follows: Initially, a global plan with no operations is generated. At each stage, the current set of sub-plans is expanded by adding the sub-plan corresponding to the most cost-effective join operation.

The cost model for A-SEGO is designed to process the join operations that have a high degree of sharing as early as possible. The degree of sharing in a specific join operation indicates the number of multi-way joins that can reuse its result. As the sharing degree increases, the effective cost decreases proportionally. For a query set \( Q \), the cost of a global execution plan denoted by \( P_Q \) may be calculated by summing the costs associated with all binary join operations involved in the query set \( Q \). The A-SEGO cost model employs a unit-time-based model [22] for a symmetric hash join. This model uses the cost statistics, i.e., the average cost and the standard deviation, of the binary join operations that have been executed at least once as part of the past global execution plan. The cost statistics are maintained in a catalog called the cost profile. Each tuple contains the cost statistics of a distinct binary join operation. Given a binary join operation \( J \), for a data stream \( D \), the cost tuple \( CS(J_D) \) consists of the following items:

- \( \text{count}(J_D) \): the number of monitoring events involved in the operation \( J_D \).
- \( \mu(\text{cost}(J_D)) \): the average cost of the operation \( J_D \).
- \( \sigma(\text{cost}(J_D)) \): the standard deviation of the operation \( J_D \).
- \( \mu(\text{cost}(J_D)^2) \): the squared average cost of the operation \( J_D \).
- \( qBit(J_D) \): the bit vector representing the queries that contain the operation \( J_D \).

The cost profiles employ a user-defined cost-bound parameter to control the number of concurrently monitored candidate subplans. As described in Section 1, an increase in the cost-bound parameter value is more likely to indicate a better global execution plan subject to the monitoring overhead for a candidate subplan. Given a global execution plan \( P_Q \) and a cost-bound parameter \( k \) for a set of queries \( Q = \{ q_1, q_2, \ldots, q_n \} \), let an \( m \)-subplan \( sp^m \) represent the evaluation order of \( m \) binary join operations \( \{ J_1, J_2, \ldots, J_m \} \) in \( P_Q \). Let \( \text{eff_cost}(sp^m) \) denote the effective evaluation cost for the \( m \)-subplan \( sp^m \). The cost-bound parameter of the \( m \)-subplan \( sp^m \), denoted \( \psi(sp^m) \), may be represented by the range \( [l^m, u^m] \), where the cost lower bound \( l^m = \mu(\text{eff_cost}(sp^m)) - k \times \sigma(\text{eff_cost}(sp^m)) \), and the cost upper bound \( u^m = \mu(\text{eff_cost}(sp^m)) + k \times \sigma(\text{eff_cost}(sp^m)) \).

Let \( SP^m_Q \) denote the set of all possible \( m \)-subplans for the query set \( Q \), and let \( sp^m_{\text{min}} = \psi(l^m_{\text{min}}, u^m_{\text{min}}) \) denote the \( m \)-subplan with the smallest cost upper bound \( u^m_{\text{min}} \) among all the \( m \)-subplans in \( SP^m_Q \). The set of candidate \( m \)-subplans, denoted by \( CSP^m \), that are traced simultaneously by A-SEGO is defined as follows:

\[
CSP^m = \{ sp^m \mid u^m_{\text{min}} > l^m, \text{and } sp^m \in SP^m_Q \}.
\]

In other words, all \( m \)-subplans having the cost-bound parameters which overlap with the cost-bound parameter of the \( sp^m_{\text{min}} \) subplans, are included in \( CSP^m \). The greater the
value of the parameter \( k \), the larger the cost-bound parameter of the subplan \( sp_{min}^m \) becomes, potentially increasing the cardinality of \( CSP \). These results indicate that the optimality of the newly generated global execution plan may be improved, although the overhead required to generate the plan can also increase. Therefore, the cardinality should be adaptively adjusted by effectively changing the parameter \( k \), considering the current system workload.

4. Adaptive Run-time Overhead Adjustment (AROA)

This section describes the proposed AROA optimization scheme, which is based on the previously proposed A-SEGO scheme. This scheme seeks to optimize the overall query execution performance by automatically adjusting the cost-bound parameter under the system environment (some system requirements and the current system workload). Section 4.1 describes the AROA system overview, and Section 4.2 illustrates how to determine the value of the cost-bound parameter adaptively to produce the most optimized execution plan while minimizing the run-time delays.

4.1. System Overview

As shown in Figure 1, the proposed scheme includes four architectural components: a query registration module, a query execution module, a run-time monitoring module, and a query optimization module. Continuous queries are registered in the query registration module. In the query execution module, the queries are evaluated according to their own execution plan. During continuous evaluation, the run-time monitoring module gathers the statistics for the costs associated with each binary join operation on the current execution plan and updates the cost statistics in the corresponding cost profile. The monitoring module examines the effectiveness of the current execution plan by measuring the cost changes. If the cost of the current plan is increased beyond the some user-defined cost threshold, the plan is considered no longer effective. Accordingly, the query optimization module invokes an optimization process to generate a new execution plan and gathers the experimental statistics describing the relationship between the cost-bound parameter and the optimization duration. Such statistics are maintained in an optimization profile. Each row of the profile maintains the pair values \((k, \hat{O})\) where \( k \) and \( \hat{O} \) denote a cost-bound parameter and an optimization duration, respectively. Here, \( k \) represents the maximum allowable value for the cost-bound parameter under the optimization duration \( \hat{O} \). When a new optimized plan is required, the optimization module first determines the maximum allowable optimization duration \( \hat{O} \) under the current system environments, then it determines the cost-bound
parameter \( k \), based on the allowable optimization duration, with reference to the optimization profile.

### 4.2. Determining Allowable Optimization Duration

Some system constraints should be considered in order to determine an allowable optimization duration \( \hat{O} \) under the current system environment, as follows:

- The data distribution may differ from the distribution that characterizes an optimized process due to the on-going characteristics of the data stream. Accordingly, the execution plan that is newly generated by the optimization process may no longer be optimal at the time it is implemented. For this reason, the optimization process must finish within a minimal period of time, during which the characteristics of a data stream are sustained. This time is called the **effective optimization time limit** and is denoted \( \xi \). The value of \( \xi \) depends mainly on the frequency with which the data distribution of a target data stream varies. The frequent change of data distribution requires the time limit \( \xi \) to be decreased.

- Strict processing time constraints are imposed due to the characteristics of the continuous queries that must be repeatedly executed at run-time. In other words, the incoming tuple of a target data stream should be processed within a specific period of time allowed under the current system environment. This time, called the **execution time limit**, is denoted by \( d \). The value of \( d \) depends mainly on the incoming rate of the tuples in a target data stream. The busy incoming rate of the tuples requires the time limit \( d \) to be decreased in order to catch up with the incoming tuples.

The terms used in the proposed scheme are summarized in Table 1.

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )</td>
<td>Cost-bound parameter</td>
</tr>
<tr>
<td>( \hat{O} )</td>
<td>Allowable optimization duration (sec)</td>
</tr>
<tr>
<td>( \xi )</td>
<td>Effective optimization time limit (sec)</td>
</tr>
<tr>
<td>( d )</td>
<td>Execution time limit (sec)</td>
</tr>
<tr>
<td>( \rho )</td>
<td>Processing time per an incoming tuple (sec/tuple)</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>The number of tuples per a second (tuples/sec)</td>
</tr>
</tbody>
</table>

Two time constraints \( \xi \) and \( d \) are supposed as system requirements. The parameters \( \rho \) and \( \lambda \) are measured periodically in the monitoring module. They are estimated on the last fixed-width past portion (time window) of the target data stream \( D \). Figure 2 shows how to calculate the the duration \( \hat{O} \). At the time \( t_1 \), the data distribution change of the stream \( D \) is detected since query processing cost measured by the monitoring module goes beyond the cost threshold. The monitoring module notifies the optimization module to begin re-optimizing the current execution plan at the time \( t_1 \). To preserve its effectiveness, the newly generated execution plan should be applied at least until the time \( t_3 (= \xi + d) \), which includes the time limit \( \xi \) and \( d \). Starting from the tuple \( d_5 \), the new plan is applied to process its succeeding tuples. The tuples \( d_1, d_2, d_3, \) and \( d_4 \) are processed by the current execution plan although their data distributions are the same as those of the tuple \( d_5 \) since the new plan does not be produced. It should be noted that the optimization module remains pending while the execution module actually processes the incoming tuples of the stream \( D \) during the duration \( \xi \). Therefore, as the duration \( \hat{O} \) is calculated, the time occupied by the execution module should be excluded from the time \( \xi \).
Figure 2. The Concept by the Optimization Duration is Calculated in the AROA Method

Algorithm. AROA approach
Input: (1) D: a target data stream
(2) c: current query processing cost
(3) c\text{thred}: processing cost threshold
(4) ξ: effective optimization time limit
(5) d: execution time limit
(6) optimization profile P: an ordered list of (k, \hat{O}) entries where k is a cost-bound parameter and \hat{O} is an optimization duration
Output: an optimization duration \hat{O}_1 and a cost-bound parameter k_1

While c = the result of periodically measuring query processing cost do
If c < c\text{thred} then
Keep the current execution plan;
Else
ρ = average processing time per an incoming tuple during the last window of D;
λ = the average number of tuples per a second during the last window of D;
\hat{O}_1 = ξ \times (1 - ρ \times λ) + d; /* calculate \hat{O}_1 */
k_1: the element k of the entry (k, \hat{O}) in P such that \hat{O} is the maximum value satisfying that \hat{O} < \hat{O}_1; /* find k_1 in P */
Generate a new execution plan based on k_1;
End if
End while

Figure 3. Algorithm of AROA Approach

Lemma 1. Given the processing time (sec.) per incoming tuple ρ and the number of incoming tuples per second λ, the occupation ratio of the execution module is ρ * λ.

Lemma 2. The allowable optimization duration \hat{O} is
\[\hat{O} = \xi \times (1 - \rho \times \lambda) + d.\] (2)

Proof. The duration \hat{O} is defined as the sum of the time d and the time occupied by the optimization module during the time ξ. According to Lemma1, the occupation ratio of the
execution module is \( \rho \ast \lambda \); therefore, the occupation ratio of the optimization module is \( 1 - \rho \ast \lambda \). □

4.3. Determining Cost-bound Parameter

Once the duration \( \hat{O} \) is determined, the cost-bound parameter \( k \) is decided with respect to the optimization profile. As described in Section 4.1, the profile stores the information about how the value of the parameter \( k \) may be decided according to the duration \( \hat{O} \). This information is gathered statistically by tracking the time required to produce the optimized plan for the specific cost-bound parameter in the monitoring module. It is stored as an ordered list of \((k, \hat{O})\) entries in the profile. This scheme maximizes the optimality of the newly generated execution plan under the current system environments because the parameter \( k \) is decided based on the duration \( \hat{O} \), which reflects the system environments.

**Example 1.** Given \( \rho = 0.001 \) (sec/tuple), \( \lambda = 600 \) (tuples/sec), \( \zeta = 3 \) (sec) and \( \delta = 1 \) (sec), the occupation ratio of the execution module is \( \rho \ast \lambda = 0.001 \ast 600 = 0.6 \). By Lemma 2, the allowable optimization duration \( \hat{O} = 3 \ast (1-0.6)+1 = 2.2 \) sec. The optimization profile then searches for the \((k, \hat{O})\) entry that satisfies the maximum value of \( \hat{O} \) among the values of \( \hat{O} \) that do not exceed 2.2 sec. The element \( k \) of the entry is the value of the cost-bound parameter. Fig. 3 shows the overview of AROA approach for finding an optimized cost-bound parameter. □

5. Experimental Studies

This section describes the experimental results supporting the performance of the proposed scheme. All experiments were implemented in C in a Linux environment using the 2.4.5 kernel and gcc 3.3.2. The experiments were executed on a Pentium 4 CPU 2.66 GHz system with 1G RAM. Various aspects of the performance were analyzed by generating several synthetic datasets. First, 30 data streams were generated consisting of 4 attributes each, including a timestamp attribute and integer-type attributes. One hundred thousand tuples were used for each stream. The values of the timestamp were filled with consecutive numbers, depending on the attribute domain size \( d_{max} \). On the other hand, each integer value was randomly chosen from the range \([1...d_{max}]\). Larger domain sizes yielded lower selectivities for a join attribute of a join operation, although these values depended on the data distribution. The input rate of each data stream was randomly selected over the range \([10...100]\) (tuples/sec). If the input rate of a specific stream was 20 tuples/sec, the timestamp of the incoming tuple was incremented by one for every 20 tuples.

All experiments described here were executed based on A-SEGO. A central goal of these studies involved comparing the proposed scheme with previous schemes from the viewpoint of optimization. The two optimization schemes examined here were based on A-SEGO; however, the optimization methods differed with respect to the determination of the cost-bound parameter \( k \). One scheme decided the parameter \( k \) using the proposed AROA method, whereas the other scheme simply used a user-defined parameter \( k \). For convenience, the former scheme is denoted AROA and the latter is denoted A-SEGO.
Figure 4. Effects of a Cost-Bound Parameter $k$

Figure 4 illustrates the effect of a cost-bound parameter $k$. For this experiment, 20 6-way join queries are evaluated on three synthetic datasets $D1$, $D2$ and $D3$ to determine the diverse characteristics of $A$-$SEGO$. The input rates and join selectivities of the stream $D1$ are randomly varied from the ranges $[10, 100]$ and $[0.001, 0.1]$ respectively. The dataset of the stream $D2$ are divided into two groups. One has low input rates within $[10, 50]$ (tuples/sec) and low join selectivities within $[0.01, 0.001]$, while the other has high input rates within $[50, 100]$ (tuples/sec) and high join selectivities within $[0.1, 0.01]$. The dataset of the stream $D3$ are also divided into two groups. One has low input rates within $[10, 50]$ (tuples/sec) and high join selectivities within $[0.1, 0.01]$, while the other has high input rates within $[50, 100]$ (tuples/sec) and low join selectivities within $[0.01, 0.001]$. All experiments for the datasets are repeatedly executed 1,000 times, presenting their average performance. All the execution plans were generated based on $A$-$SEGO$ approach. In order to measure the effect of a cost-bound parameter $k$, the cost function $\hat{E}(k)$ is defined as follows:

$$\hat{E}(k) = \frac{\text{cost}_k - \text{cost}_{opt}}{\text{cost}_{opt}}$$

(3)

where $\text{cost}_k$ denotes the cost of an execution plan generated by $A$-$SEGO$ and $\text{cost}_{opt}$ denotes the cost of an ideal optimal plan.

$\hat{E}(k)$ measures the relative effectiveness of $A$-$SEGO$ approach for the ideal optimal plan that cannot be implemented at run-time. According to the definition, the smaller the value of $\hat{E}(k)$, the closer to optimality is the execution plan generated by $A$-$SEGO$. As shown in this figure, the value of $\hat{E}(k)$ decreases as that of $k$ increases; thus, the optimality of $A$-$SEGO$ is increased. It means that the volume of candidate sub-plans decided by the parameter $k$ has the critical effect on the query performance. Especially, it is interesting that the performance of $A$-$SEGO$ for the stream $D3$ is less than that for the other synthetic datasets. It means that, although a join operation with low join selectivity is scheduled earlier, it does not guarantees better performance since its result volume is still large due to its high operand input rate in the stream $D3$. Consequently, it can be inferred that the data distribution characteristics of the stream $D3$ increase the risk of generating a sub-optimal global plan.

Figure 5 illustrates the trade-off between the processing costs of an execution plan and the optimization time (duration). As shown, an increase in $k$ increased the time required to generate a new execution plan, whereas the processing costs associated with the plan
decreased. As $k$ increased beyond a specific threshold ($k = 1.5$ in this experiment), the optimization time is increased rapidly while the decrease of the processing cost by the new plan is trivial. It emphasizes the importance of selecting the parameter $k$ since the simultaneous tracking of multiple sub-plans can incur a high overhead cost.

![Figure 5. The Trade-Off Between the Processing Cost and the Optimization Time](image)

Figure 5. The Trade-Off Between the Processing Cost and the Optimization Time

![Figure 6. Effects of the Total Input Rate](image)

Figure 6. Effects of the Total Input Rate

Figure 6 compares the A-SEGO and AROA methods by varying the total input rate of the source data streams. The magnitude of the run-time delay varied depending on the evaluation of multiple continuous queries of the incoming tuples in the streams. Figure 4(a) and Figure 4(b) shows the average and maximum run-time delays, respectively. The input rate represents the system workload. In this experiment, 20 queries were employed, each of which included seven join predicates. The fluctuations in the data distributions over the stream were simulated by creating 20 distinct sub-datasets, the tuples of which were processed in turn. Each sub-dataset included 500 tuples per source stream, and the domain size of the join attributes varied randomly over the range [10, 1000].

A-SEGO employed a user-defined constant value as the cost-bound parameter $k$, i.e., $k=1$, whereas AROA adaptively controlled the parameter $k$ according to the total input
rate (system workload). As the input rate increased, the values of the parameter \( k \) decreased because the optimization duration shortened, as described in Section 4.2. As shown in this figure, AROA outperformed A-SEGO \((k=1)\) overall. As the input rate increased (i.e., the system workload became high), the performance gap between the two methods increased, indicating that the method by which a cost-bound parameter is determined can affect the query performance. The AROA optimization method was more effective than the A-SEGO method, especially for a system with a heavy workload. The continuous query results incurred a smaller run-time delay response by adjusting the cost-bound parameter in the AROA scheme.

Figure 7 illustrates the effects of the two limit parameters discussed in Section 4.2: the effective optimization time limit \( \xi \) and the execution time limit \( d \). This experiment examined a set of 50 six-way join queries. The dataset used in the experiments described in Fig. 4 were employed here. Figure 5(a) shows that the run-time delay, a yardstick for query performance, decreased as the limit \( \xi \) increased. An increase in the limit \( \xi \) increased the cost-bound parameter \( k \), which improved the chances of generating a more optimized execution plan; however, this figure also shows that the performance worsened once the value of the limit \( \xi \) reached a threshold (8 sec in this case). These results indicated that a newly generated plan may no longer be effective when it replaces an old plan because the characteristics of the experimental dataset can change if too much time has elapsed during the optimization search. Fig. 5(b) shows the effects of the limit \( d \) on the optimization scheme performance. As described in Section 4.2, the value of \( d \) was proportional to the incoming rate of the tuples in a source data stream. Therefore, an increase in the limit \( d \) indicated an increase in the system workload, which in turn increased the run-time delay and worsened the query performance.

6. Conclusions

A novel AROA method based on the previously proposed schemes k-EGA and A-SEGO, which adaptively generate an optimized execution plan for multiple continuous multi-way join queries, was proposed as a means for adaptively adjusting the cost-bound parameter according to the current system environment. The AROA method calculated the allowable optimization duration under the current system workload, and the cost-bound parameter was adaptively determined based on the calculated
optimization duration. The relationship between the cost-bound parameter and the optimization duration was measured experimentally, based on monitoring of the actual execution results. The AROA scheme was found to perform better than the k-EGA and A-SEGO schemes by providing an empirical method for effectively deciding the value of the cost-bound parameter. This technique can significantly improve the query performance. A series of experiments was used to verify that the AROA scheme could reduce the run-time delay relative to the A-SEGO scheme.

Acknowledgements

This research was supported by Basic Science Research Programs (NRF-2012R1A1A2009170) through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and Future Planning.

References


Authors

Hyun-Ho Lee received the B.S., M.S. and Ph.D. degrees in the Department of Computer science at Yonsei Univ., S. Korea in 2007. He is currently a associate professor in Yeonsung Univ., S. Korea. Also, he works as an auditor in Software&Contents Laboratory.

Hong-Kyu Park received the B.S., M.S. and Ph.D. degrees in the Department of Computer science at Yonsei Univ., S. Korea in 2012. He is currently a research faculty in the Media Solution Center(MSC) of Samsung Electronics Co. Ltd., S. Korea.

Jin-Chul Park received the B.S. degree in Computer Science and Engineering from Hanyang Univ., S. Korea in 2006. Also, he has completed M.S./Ph.D. integration degree at Department of Computer Science & Engineering at Hanyang Univ. in 2013. He can be contacted at: Department of Computer Science and Engineering.

Won-Suk Lee received the B.S. degree in Computer Engineering from Boston University, Boston and the M.S. and Ph.D. degrees in Electrical and Computer Engineering from Purdue University, West Lafayette, IN. He is currently a professor of Department of Computer science at Yonsei Univ., Korea. Also, he works as a represen-tative in Software&Contents Laboratory.
Kil-Hong Joo received the M.S. and Ph.D. degree in Computer Science from Yonsei University, Seoul, Korea, in 2000 and 2004. He is currently a professor of Department of Computer Education at Gyeongin National University of Education, Korea. His current interests include mining data streams, smart learning and Flipped classroom.