Advanced Science and Technology Letters
Vol.62 (Sensor 2014), pp.70-73
http://dx.doi.org/10.14257/astl.2014.62.18

Run-time Overhead Reduction for Multiple Continuous Query Processing

Hyun-Ho Lee[1], Hong-Kyu Park[2], Jin-Chul Park[3], Won-Suk Lee[4], Kil-Hong Joo[5]*

[1] Yonseong University, Korea, hhlee@yeonsung.ac.kr,
[2] Samsung Electronics Co. Ltd., Korea, gladiator11@hanmail.net,
[3] Hanyang University, Korea, jin1chul@nate.com,
[4] Yonsei University, Korea, leewo2001@gmail.com,
[5] Gyeonin National University of Education, Korea, khjoo@ginue.ac.kr

Abstract. The previously proposed scheme, like k-EGA and A-SEGO, employs a user-defined cost-bound parameter for controlling the number of monitored subplans. This paper proposes a new scheme, Adaptive Run-time Overhead Adjustment(AROA), which adaptively determines the value of a cost-bound parameter based on the system environment. Experimental verification shows that AROA outperforms A-SEGO.

Keywords: data stream, join optimization, AROA, cost-bound parameter.

1 Introduction

Due to the strict time constraints of the continuous queries in a stream environment, query optimization processes are essential. Previous studies have proposed a multi-way join algorithm, k-EGA [1,2,3], for the adaptive production of an optimized execution plan for a single continuous multi-way join over a data stream [4]. 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 [5]. 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.

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 propability of finding an optimal global plan is increased but the overhead of monitoring the candidate subplans is also increased. This overhead

* Corresponding Author

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. 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.

2 Adaptive Run-time Overhead Adjustment(AROA)

AROA 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. Based on the monitoring results, 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.

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

· 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 ξ.

· Strict processing time constraints are imposed due to the characteristics of the continuous queries that must be repeatedly executed at run-time. This time, called the *execution time limit*, is denoted by $đ$.

Lemma 1. The allowable optimization duration $\hat{O}$ is

$$\hat{O} = \xi \times (1- \rho \times \lambda) + \eth. \quad (1)$$

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

Algorithm. ***AROA approach***

```
Input: (1) $D$: a target data stream
    (2) $c$: current query processing cost
    (3) $c_{thred}$: processing cost threshold
    (4) $\xi$: effective optimization time limit
    (5) $\eth$: 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 $c < c_{thred}$ then
    Keep the current execution plan;
  Else
    $\rho$ = average processing time per an incoming tuple during the last window of $D$;
    $\lambda$ = the average number of tuples per a second during the last window of $D$;
    $\hat{O}_1 = \xi \times (1- \rho \times \lambda) + \eth$; /* 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
```

Fig. 1. Algorithm of AROA approach

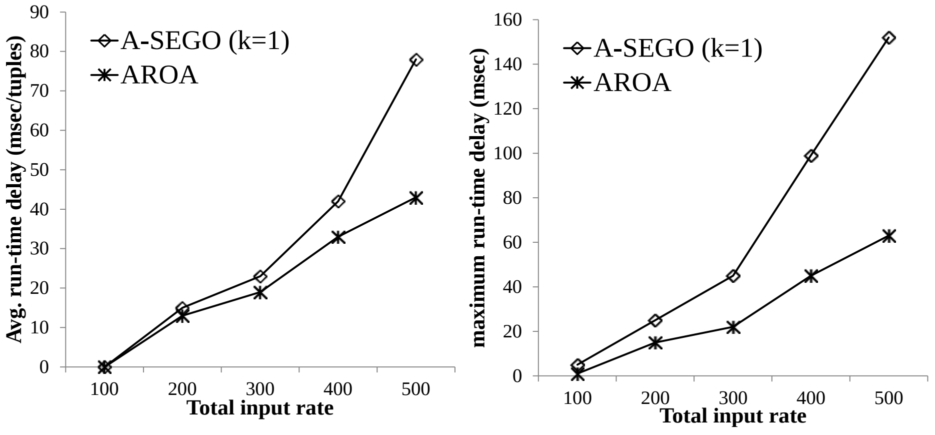

Fig. 2. Effects of the total input rate (A-SEGO *vs.* AROA)

3 Experimental Studies

Figure 2 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. 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]. As shown in this figure, the AROA was more effective than the A-SEGO, especially for a system with a heavy workload.

4 Conclusions

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.

Acknowledgments. 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

1. S. Madden, M. Shah, J. M. Hellerstein, and V. Raman: Continuously Adaptive Continuous Queries over Streams, SIGMOD, 49 – 60 (2002)
2. R. Avnur and J. Hellerstein: Eddies: Continuously adaptive query processing, SIGMOD, 261-272 (2000)
3. Y. Simmhan, B. Cao, M. Giakkoupis, and V. K. Prasanna: Adaptive rate stream processing for smart grid applications on clouds, ScienceCloud. ACM, New York, NY, USA, 33-38 (2011)
4. H. K. Park, and W. S. Lee: Adaptive Continuous Query Reoptimization over Data Streams, IEICE Transactions 92 (7), 1421-1428 (2002)
5. H. K. Park, and W. S. Lee: Adaptive Optimization for Multiple Continuous Queries, Data & Knowledge. Engineering, http://www.sciencedirect.com/science/article/pii/S0169023X11001054, (online publishing)