J Intell Inf Syst (2012) 39:317–334
DOI 10.1007/s10844-011-0192-1

Adaptive two-level optimization for selection predicates of multiple continuous queries

Hyun-Ho Lee · Won-Suk Lee

Received: 1 April 2011 / Revised: 30 November 2011 / Accepted: 5 December 2011 /
Published online: 22 January 2012

Abstract A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. Query processing for such a data stream should also be continuous and rapid, which requires strict time and space constraints. In order to guarantee these constraints, we have proposed a new scheme called an Attribute Selection Construct (ASC) for an attribute of a data stream in our previous study (Lee and Lee, Information Sciences 178:2416–2432, 2008). As its optimization technique, this paper proposes the new strategy that determines the evaluation order of multiple ASC's for a given query set at two different levels—macro and micro levels. Based on the two levels, it also proposes two different strategies—macro-sequence and hybrid-sequence—that find the optimized full evaluation sequence of all the ASC's. In addition, it provides the adaptive strategy that periodically rearranges the evaluation sequence of multiple ASC's. The performance of the proposed technique is verified by a series of experiments.

Keywords Data stream · Multiple continuous queries · Selection predicate · ASC · Macro level · Micro level · Macro sequence · Hybrid sequence · Adaptive optimization

1 Introduction

A data stream is defined as a massive unbounded sequence of data elements continuously generated at a rapid rate (Babcock et al. 2002; Motwani et al. 2003).

H.-H. Lee (✉)
Department of Non-commissioned officers, Anyang Science University,
San39-1 Anyang3-Dong Manan-Gu, Anyang-Si, Gyeonggi-Do, Korea
e-mail: hhlee@ianyang.ac.kr

W.-S. Lee
Department of Computer Science, Yonsei University, 134 Sedaemoongu,
Shinchondong, Seoul, Korea
e-mail: leewo@database.yonsei.ac.kr

Accordingly, a registered query in a data stream management system (DSMS) is called a *continuous query*. It should be executed continuously rather than once on demand, producing its results whenever a new tuple of a target data stream arrives (Abadi et al. 2003; Avnur and Hellerstein 2000; Chen et al. 2002). Research activities on data streams are motivated by emerging applications involving massive datasets such as customer click streams, multimedia data, retail chain transactions and network intrusion detection system (NIDS). In these fields, one of the main research issues is *message brokers* that classify data by some criteria and send them to proper destinations. The classification criteria are implemented by some continuous queries. They should be evaluated in real-time, which requires strict time and space constraints. Since a number of continuous queries are registered together in advance, it is more efficient to evaluate multiple queries collectively by sharing the common constraints of the queries, as already proposed in most DSMS's (Chandrasekaran and Franklin 2002; Chen et al. 2000; Madden et al. 2002; Sharaf et al. 2007).

We have proposed a new structure called an *attribute selection construct (ASC)* for the efficient evaluation of selection predicates (Lee and Lee 2008). Given a set of continuous queries, an attribute of a base data stream is defined as a *p-attribute* (*participant attribute*) if it is employed to express at least one selection predicate. An *ASC* is constructed for each p-attribute and it contains the encoded information of those selection predicates that are imposed to its corresponding p-attribute. Based on the constraining constant values of the selection predicates, the entire domain of a p-attribute is subdivided into a number of *disjoint regions*. For every region, the *ASC* of the p-attribute maintains pre-computed results for all the queries. The results indicate which queries satisfy an incoming tuple if its p-attribute value falls within the region. An *ASC* can be completely built at compile-time because only the selection predicates of the queries are required to build it. This feature improves run-time efficiency which is important in timely fashioned stream environment.

Based on the *ASC* scheme, this paper proposes a new adaptive two-level optimization technique. Given a set of continuous queries, a tuple of a data stream can be dropped when it does not satisfy any of the queries. In case of a detection system such as NIDS, tuple filtering capability should be considered significantly because most of normal ones should be filtered out by the system. In order to minimize the run-time overhead of query evaluation, it is very important to filter out such an unmatched tuple as early as possible (Babu et al. 2004). Among the multiple *ASC*'s constructed for the queries, the filtering capability of each *ASC* is also different from one another. Furthermore, within an *ASC* its regions also have different filtering capabilities. Therefore, the evaluation order of the *ASC*'s can significantly influence the overall performance of query evaluation. From this viewpoint, the proposed method determines the evaluation order of multiple *ASC*'s at two different levels. One is only *ASC*'s level (*macro level*). The other is the combination level of *ASC* and its regions (*micro level*). Based on the two levels, this paper proposes two different strategies—*macro-sequence* and *hybrid-sequence*. A macro sequence finds the full evaluation order of all the *ASC*'s, considering only their overall filtering capabilities at macro level. Meanwhile, a hybrid sequence considers the respective filtering capabilities of the regions of an *ASC* at micro level, as well as its overall filtering capability. Also, this paper includes an adaptive strategy which dynamically rearranges the current evaluation sequence, capturing the run-time tuple dropping ratio of the sequence periodically. As a user-defined parameter,

a *rearrangement threshold* μ is introduced to indicate the maximum allowable ratio that the inefficiency of the current evaluation sequence can be sustained by.

Contributions The contributions of this paper are summarized as follows:

- Based on the previous proposed scheme *ASC* (Lee and Lee 2008), it proposes the new techniques determining the evaluation sequence of *ASC*'s at two different levels—*macro* and *micro* levels—for multiple target queries over a data stream.
- Considering macro and micro levels, it provides two different strategies that find the optimized full evaluation sequence of multiple *ASC*'s: a *macro-sequence* and a *hybrid-sequence*. A macro-sequence determines the sequence at only the macro level and a hybrid-sequence does the sequence at both the macro and micro levels.
- Due to the selectivity change of selection predicates over a data stream, it also provides the adaptive optimization strategy that periodically rearranges the evaluation sequence of multiple *ASC*'s.

Paper outline Section 2 presents related works and Section 3 (preliminary section) briefly introduces how to construct and evaluate the *ASC* proposed in our previous paper (Lee and Lee 2008). Sections 4 and 5 illustrate how to find the optimized evaluation sequence of *ASC*'s at two different levels, and how to evaluate and rearrange it at run-time, respectively. In Section 6, the performance of the proposed method is analyzed through a series of experiments. Finally, Section 7 presents our conclusions.

2 Related work

Some studies have proposed a grouping-based method for sharing the common constraints of continuous queries. *CACQ* (Madden et al. 2002) employs a predicate index for each distinct attribute and maintains various data structures reflecting the characteristics of comparison operators such as an equality hash-table and a greater-than(less-than) tree. *PSoup* (Chandrasekaran and Franklin 2002) uses a red-black tree based on an IBS-tree (Interval Binary Search tree) (Hanson et al. 1990) for each distinct attribute in order to index all the constraining constants of selection predicates. In case of a non-equal comparison, both of the predicate index of *CACQ* and the red-black tree of *PSoup* should be traversed sequentially for its specified range, which can degrade the performance of query evaluation considerably if it is highly selective. (Wu et al. 2006) uses a query index with a series of hierarchical *CEI*'s *(Containment-Encoded Intervals)* for each distinct attribute. Similarly to the disjoint region of the proposed *ASC*, a *CEI* is produced by exclusively dividing the domain of a corresponding attribute according to the constraining constants of its selection predicates. However, in order to find a set of satisfied queries, more than one *CEI* which contains the corresponding attribute value of an incoming tuple should be searched in a cascade. Meanwhile, in *ASC*, only one region is searched because each region contains the evaluation results for all the queries. In general, *ASC* requires more space than *CEI*'s. However, with regard to search complexity, *ASC* is simpler than *CEI*'s. In other words, compared to (Wu et al. 2006), our approach

focuses on reducing evaluation cost rather than storage cost. There are some efficient filtering algorithms in *publish/subscribe* systems (Fabret et al. 2001). The proposed scheme of (Fabret et al. 2001) groups subscriptions based on their size and common conjunction of equality predicates, and uses multi-attribute hash indices so several subscription attributes can be evaluated using a single comparison. However, as the number of subscription attributes is increased, multi-attribute hash indices should be maintained more complicatedly. Moreover, not only they cannot benefit from short-cut operations in the conjunctive form of selection predicates but also they need extra operations for inequality predicates.

Despite of run-time overhead, several adaptive optimization strategies are also proposed. *Eddies* (Avnur and Hellerstein 2000) creates a selection module for an attribute that is used to express a selection predicate. The execution order of multiple selection modules is decided based on their selectivities. It is also changed adaptively by tracking the selectivities over all the tuples the module has processed recently. Bizarro et al. (2005) has proposed *CBR (Content-Based Routing)*, which extends Eddies to support different routes for a single data stream. *CBR* uses adaptive algorithms that partition input data based on statistical properties, and efficiently route individual tuples through customized plans based on their partition. These adaptive methods can degrade the performance of query evaluation especially when the number of selection modules is large *or* data distribution of incoming tuples is frequently changed. *STREAM* (Widom and Babu 2001) uses the A-greedy algorithm which monitors the on-going selectivity statistics of various partial evaluation sequences for the selection predicates at run-time. If the current order is not optimal, it is rearranged adaptively. However, it can only be applied to a single continuous query. In (Munagala et al. 2006), a shared execution strategy is proposed for the purpose of optimizing multiple continuous queries. For a specific incoming tuple, among the shared filters that are not evaluated yet, the next filter to be evaluated is chosen by cost-based analysis at run-time. Since the unit of evaluation scheduling is an individual filter, its run-time complexity can be rapidly increased when the number of filters is large. Furthermore, there is no facility that controls the trade-off between query performance and optimization overhead. (Wang et al. 2006) has proposed a query index tree based on decision tree. All kinds of predicates indices on p-attributes are integrated into a single index tree. The optimization point of this scheme is how to select dividing p-attributes during the construction of the tree. For this purpose, (Wang et al. 2006) uses either of *Information Gain (IG)* or *Estimated Time Cost (ETC)*. However, since the evaluation sequence of p-attributes is decided by the internal structure of an index tree, it is hard work to optimize the evaluation sequence adaptively. In fact, the index tree should be reconstructed for adaptive optimization.

3 Preliminaries

3.1 ASC: attribute selection construct

As described in Section 1, an *ASC* stores the pre-computed matching results of all the regions of the corresponding p-attribute. It is constructed for each p-attribute and is formally defined as follows.

Definition 1 (Attribute Selection Construct)
Given a set of continuous queries $Q = \{q_1, q_2, \ldots, q_k\}$ registered to a relational stream $D(A_1, A_2, \ldots, A_n)$, let $A_p(Q) \subseteq \{A_1, A_2, \ldots, A_n\}$ denote the set of p-attributes for the query set Q. An attribute selection construct $ASC(A_i)$ for a p-attribute $A_i \in A_p(Q)$ with m distinct constraining constant values has the following entries:

- **Query-usage bitmap (qub[1..k])** indicates whether the p-attribute A_i is employed in the selection predicates of the jth query $q_j \in Q(1 \leq j \leq k)$ or not. In other words, if $A_i \in A_p(\{q_j\})$, $qub[j] = 1$. Otherwise, $qub[j] = 0$.
- **Region array (ra[1..$2m+1$])** has $(2m + 1)$ entries which correspond to $(2m + 1)$ distinct regions. The rth region $ra[r]$ $(1 \leq r \leq 2m + 1)$ maintains the following two fields:
  - **Region identifier (rif)** indicates the range that the region covers, explicitly *or* implicitly. If the rth region $ASC(A_i).ra[r]$ is the constant region that only includes the constant C, then $ASC(A_i).ra[r].rif = C$. Otherwise, $ASC(A_i).ra[r].rif = null$, implying that $ASC(A_i).ra[r-1].rif < ASC(A_i).ra[r].rif < ASC(A_i).ra[r+1].rif$.
  - **Query-result bitmap (qrb[1..k])** stores the pre-computed matching result of a region. If any tuple whose p-attribute A_i's value falls within the region $ASC(A_i).ra[r]$ cannot satisfy the jth query $q_j \in Q$, then the jth bit of the bitmap is set to 0 i.e., $ASC(A_i).ra[r].qrb[j] = 0$. Otherwise, the jth bit is set to 1. i.e., $ASC(A_i).ra[r].qrb[j] = 1$.

If the jth query q_j does not have any selection predicate for a p-attribute A_i i.e., $ASC(A_i).qub_j = 0$, then its corresponding query-result bit $ASC(A_i).ra[r].qrb[j]$ should be set to 1 in all the regions of $ASC(A_i)$ because the matching result of the query q_j cannot be determined by the p-attribute A_i.

Example 1 In Fig. 1, the query-result bitmap $ASC(A_2).ra[2].qrb = 1110$ indicates that only the query q_4 is not satisfied if the A_2's attribute value of an incoming tuple is equal to *20*. In this region, the queries q_1 and q_3 are also considered to be satisfied because the p-attribute A_2 is not used in its selection predicates.

3.2 Run-time evaluation of ASC's

Given a set of continuous queries $Q = \{q_1, q_2, \ldots, q_k\}$, the *ASC*'s of their p-attributes $A_p(Q)$are processed for every incoming tuple one by one in sequence. A *global-result bitmap* denoted by a *GRB[1..k]* is introduced to accumulate the intermediate matching result of each *ASC* in the course of query evaluation. Initially, all the bits of the bitmap *GRB* are set to 1's, assumed that all of the queries in Q satisfy an incoming tuple t_D of the data stream. The ASC of a p-attribute $A_i ASC(A_i)$ is evaluated as follows: Among the disjoint regions of $ASC(A_i).ra$, the one that includes the A_i's attribute value of the incoming tuple t_D is identified first by binary-searching, based on the constant values of its region-identifiers. Subsequently, let the identified region be $ASC(A_i).ra[r]$. Its query-result bitmap $ASC(A_i).ra[r].qrb$ is *bitwise ANDed* with the global result bitmap *GRB*. The result of this operation is reassigned to *GRB*. If the updated result of *GRB* is 0, the incoming tuple t_D is dropped immediately

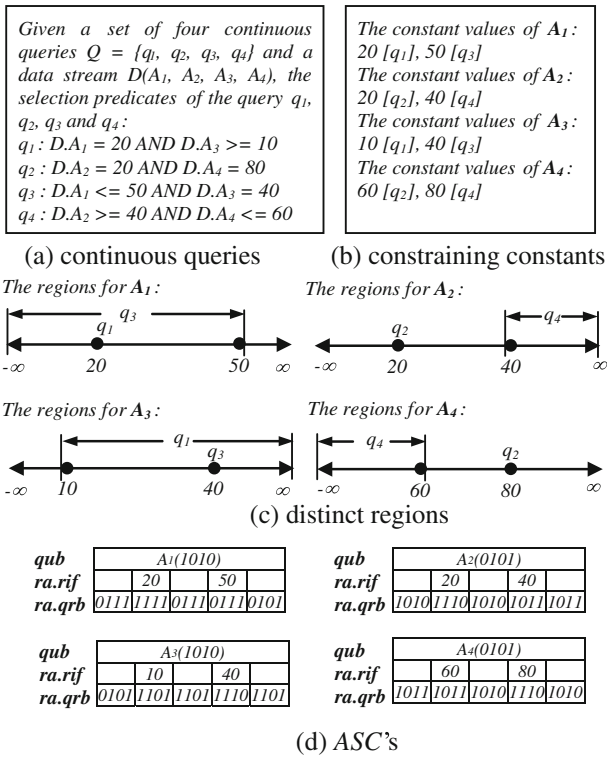

Fig. 1 The constructing process of *ASC*'s

even if there are some *ASC*'s left to be evaluated. Otherwise, the matching process is continued by evaluating the next *ASC*. When all the *ASC*'s have been processed successfully, the queries satisfied by the incoming tuple t_D are identified.

4 Evaluation sequence

Given a set of continuous queries Q, if the number of p-attributes in the query set Q is more than one i.e., $|A_p(Q)| > 1$, the evaluation sequence of their corresponding *ASC*'s can significantly affect the query performance due to the difference in the filtering capabilities of the p-attributes. Therefore, finding an optimized evaluation sequence is very important. The proposed method determines the evaluation order of *ASC*'s at two different levels. The first determines the order based on the overall filtering capability of each *ASC* by averaging the filtering capabilities of all of its regions. The second determines it based on the individual capability of each region of an individual *ASC*. Given two *ASC*'s, an evaluation order identified by the first level is called a *macro-arrow*, whereas an evaluation order identified by the second level is called a *micro-arrow*. A micro-arrow starts from a specific region of one *ASC* and ends to the other *ASC*. This paper proposes two different strategies that find the overall evaluation sequence of multiple *ASC*'s. One determines the evaluation sequence only by a sequence of macro-arrows. It is called a *macro-sequence*. The other determines the evaluation sequence by a sequence of macro/micro arrows. It

is called a *hybrid-sequence*. In a hybrid-sequence, a macro-arrow is used when the filtering capability of a specific region of an *ASC* is much more selective. While the evaluation order of a macro-sequence is fixed, that of a hybrid-sequence may be different according to the attribute values of an incoming tuple as in (Bizarro et al. 2005). In order to find the evaluation sequence at run-time, a monitoring module should be kept apart from the executor like the *streaMon* in *STREAM* (Widom and Babu 2001). It identifies the optimized sequence during executing continuous queries, based on the selectivities of *ASC*'s.

4.1 Extension of ASC

To implement the proposed evaluation sequence, the structure of *ASC* is slightly extended. Based on Definition 1, a new entry called a *candidate-arrow bitmap* is added to the region array of *ASC*. It is defined as follows:

– **Candidate-arrow bitmap** ($cab[1..|A_p(Q)|]$) indicates a set of *ASC*'s which are candidates for the next *ASC* to be evaluated. For those tuples whose values of the p-attribute A_i are in the region $ASC(A_i).ra[r]$, $ASC(A_j)(j \neq i)$ is a candidate of $ASC(A_i)$ if and only if it can fail at least one query in Q right after evaluating $ASC(A_i)$. In other words, if $ASC(A_i).ra[r].qrb \;\&\; ASC(A_j).qub \neq 0$, $ASC(A_i).ra[r].cab[j] = 1$. A micro-arrow from the region $ASC(A_i).ra[r]$ is established to one of those *ASC*'s whose corresponding bits of $ASC(A_i).ra[r].cab$ are 1's through monitoring their selectivities at run-time.

Figure 1 shows how the *ASC*'*s* of conjunctive continuous queries are constructed. The constant values of the selection predicates used in the four queries are arranged by their p-attributes in Fig. 1b. Subsequently, in Fig. 1c, the domain of each p-attribute is divided into a set of disjoint regions based on the constant values. Since the basic components of an *ASC* are constructed at compile-time, maintaining *ASC*'s causes nearly negligible run-time overhead.

Example 1 In Fig. 2, the candidate-arrow bitmap $ASC(A_2).ra[3].cab = 1010$ indicates that a micro-arrow from this region can be destined to either $ASC(A_1)$ or $ASC(A_3)$. A micro-arrow whose destination is $ASC(A_4)$ is not established because $ASC(A_2).ra[3].qrb \;\&\; ASC(A_4).qub = 0$.

Example 2 Figure 3 illustrates how the queries in Fig. 1 are evaluated by an evaluation sequence of macro-arrows $ASC(A_1) \rightarrow ASC(A_2) \rightarrow ASC(A_3) \rightarrow ASC(A_4)$. For the tuple t_2, the matched region of $ASC(A_1)$ is the last region whose boundary is $(70, \infty)$ because $t_2[A_1] = 70$. The query-result bitmap $ASC(A_1).ra[5].qrb = 0101$

Fig. 2 The extension (candidate-arrow bitmap) of Fig. 1

qub	$A_1(1010)$				
ra.rif		20		50	
ra.qrb	0111	1111	0111	0111	0101
ra.cab	0111	0111	0111	0111	0101

qub	$A_2(0101)$				
ra.rif		20		40	
ra.qrb	1010	1110	1010	1011	1011
ra.cab	1010	1011	1010	1011	1011

qub	$A_3(1010)$				
ra.rif		10		40	
ra.qrb	0101	1101	1101	1110	1101
ra.cab	0101	1101	1101	1101	1101

qub	$A_4(0101)$				
ra.rif		60		80	
ra.qrb	1011	1011	1010	1110	1010
ra.cab	1110	1110	1010	1110	1010

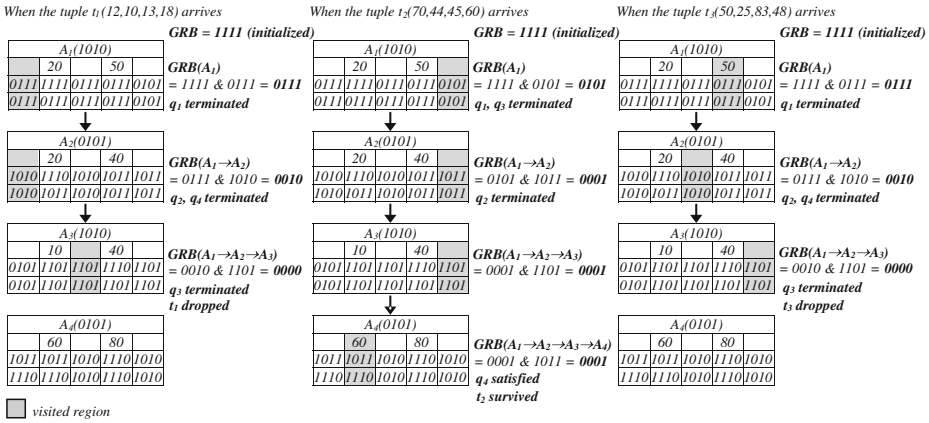

Fig. 3 Run-time evaluation of *ASC*'s

is *bitwise ANDed* with the global-result bitmap *GRB* whose bits are initialized to all 1's. The result of this operation makes $GRB = 0101$. This means that the queries q_1 and q_3 are terminated because they do not satisfy the tuple t_2. In the same way, $ASC(A_2)$, $ASC(A_3)$ and $ASC(A_4)$ are also processed one by one. Since the value of the bitmap *GRB* is finally set to *0001*, only the query q_4 satisfies the tuple t_2. For the tuples t_1 and t_3, all of the four queries turn out to be unsatisfied after processing the third *ASC* $ASC(A_3)$. Consequently, both of t_1 and t_3 are dropped and the remaining ASC $ASC(A_4)$ is not evaluated.

4.2 Macro-sequence

4.2.1 Minimal cover set

The overall performance of evaluating the queries depends on how early unmatched tuples are dropped because all the queries are collectively processed in the proposed scheme. An incoming tuple of a data stream can be dropped only after the fact that it cannot satisfy any query in a query set Q is determined. If a specific query $q \in Q$ does not have any selection predicate for a p-attribute A_i, the *ASC* of the p-attribute A_i cannot be employed to determine the matching result of the query q. Therefore, for efficient tuple dropping, it is important to first evaluate those p-attributes that can collectively provide the complete matching result of every query in Q. Such a set of p-attributes is called a *cover set*. A cover set with the smallest cardinality is defined to be a *minimal cover set.* A tuple cannot be dropped until all the p-attributes of a minimal cover set are evaluated. A minimal cover set is formally defined, as follows.

Definition 2 (Minimal cover set)

Given a set of continuous queries $Q = \{q_1, q_2, \ldots, q_k\}$ registered to a data stream $D(A_1, A_2, \ldots, A_n)$, let V be a subset of p-attributes i.e., $V \subseteq A_p(Q)$. If all the *query-usage bitmaps* of the *ASC*'s corresponding to the p-attributes in V are *bitwise ORed* to all 1's, then the set V is called a *cover set CS(Q)* for the queries in Q. A cover set *CS(Q)* with the smallest cardinality is a *minimal cover set MCS(Q)*.

Example 3 In Fig. 3, $MCS(Q) = \{A_1, A_2\}$ or $\{A_3, A_4\}$ because $ASC(A_1).qub$ & $ASC(A_2).qub = 0$ and $ASC(A_3).qub$ & $ASC(A_4).qub = 0$.

4.2.2 Finding macro-sequence

A macro-sequence is complete and unique because it arranges the *ASC*'s of all the p-attributes for a given query set according to their average filtering capabilities. Based on the concept of a *minimal cover set*, a macro-sequence for a query set Q can be found by two phases. In the first phase, a minimal cover set *MCS(Q)* is identified, and the sequence of its p-attributes is also identified simultaneously during finding *MCS(Q)*. In the second phase, a complete macro-sequence is identified by arranging the p-attributes excluded from *MCS(Q)*.

To find *MCS(Q)*, while expanding a p-attributes sequence ρ in a greedy manner, those queries whose matching results are already determined by the previous p-attributes of the sequence ρ are excluded from the query set Q temporarily. For a set of k continuous queries $Q = \{q_1, q_2, \ldots, q_k\}$ and a p-attribute $A_i \in A_p(Q)$, the corresponding attribute value of an incoming tuple should be evaluated for all the queries in Q in order to decide whether it can satisfy each of the queries or not. Let a term *evaluation instance* denote an evaluation task of an individual query for an incoming tuple on a specific *ASC*. Therefore, for every incoming tuple, k individual evaluation instances should be taken place. The *conditional selectivity of a p-attributes sequence* is formally defined as follows.

Definition 3 (Conditional selectivity)
Given a set of k continuous queries $Q = \{q_1, q_2, \ldots, q_k\}$ registered to a data stream $D(A_1, A_2, \ldots, A_n)$, let a p-attributes sequence ρ be a partial sequence of p-attributes, which is not a minimal cover set *MCS(Q)* yet. And it is supposed that the selection predicates for the p-attributes of the sequence ρ cover only w queries in $Q(w < k)$. To find the complete evaluation sequence of *MCS(Q)*, the sequence ρ is repeatedly expanded by appending one of the remaining p-attributes in $A_p(Q)$. When a p-attribute $A_v \notin \rho$ is appended to the sequence ρ, the conditional selectivity $s_\tau(\rho \rightarrow A_v|\rho)$ of the expanded sequence $\rho \rightarrow A_v$ for the sequence ρ in a fixed period τ is defined as follows:

$$s_\tau(\rho \rightarrow A_v|\rho) = \frac{EI_\tau(\rho \rightarrow A_v|\rho)}{(k-w) \times T_\tau}. \quad (1)$$

where T_τ denotes the total number of tuples generated in the period τ, and $EI_\tau(\rho \rightarrow A_v|\rho)$ denotes the number of evaluation instances *(EI)* successfully passed by the sequence $\rho \rightarrow A_v$ for those $(k-w)$ remaining queries whose results are not determined by the sequence ρ.

This conditional selectivity is employed until the currently expanding evaluation sequence of p-attributes becomes a minimal cover set *MCS(Q)*. The number of queries covered by a partial sequence ρ can be traced efficiently by the query-usage bitmaps of the *ASC*'s in the sequence ρ. It can be obtained by counting the number of 1's in the result of a *bitwise OR* operation on all the query-usage bits of the *ASC*'s. Furthermore, the value of $EI_\tau(\rho \rightarrow A_v|\rho)$ can also be efficiently found at run-time by the global result bitmap *GRB* as follows. Let $Q_p(\rho)$ be the set of queries covered by the sequence ρ. Since every query $q \in Q_p(\rho)$ is excluded from computing the conditional selectivity of the sequence $\rho \rightarrow A_v$, if T_τ tuples $\{t_1, \ldots, t_T\}$

are generated in a period τ, $EI_\tau(\rho \rightarrow A_v|\rho)$ for a query set $Q = \{q_1, q_2, \ldots, q_k\}$ is expressed as follows:

$$EI_\tau(\rho \rightarrow \mathrm{A_v}|\rho) = \sum_{i=1}^{|\tau|} \eta(\rho \rightarrow A_v, t_i). \quad (2)$$

where $\eta(\rho \rightarrow A_v, t_i)$ denotes the number of 1's in those bit positions of *GRB[1..k]* that are corresponding to the queries in $Q - Q_p(\rho)$ after the *i*th tuple t_i is evaluated for the sequence $\rho \rightarrow A_v$.

After identifying *MCS(Q)*, a complete macro-sequence is determined by extending the evaluation sequence of *MCS(Q)* one by one. Among all the candidate sequences, the one with the highest tuple dropping ratio is chosen. The tuple dropping ratio of a p-attributes sequence is formally defined as follows.

Definition 4 (Tuple dropping ratio)
Given a set of continuous queries $Q = \{q_1, q_2, \ldots, q_k\}$ registered to a data stream $D(A_1, A_2, \ldots, A_n)$ and a p-attributes sequence $\rho = A_x \rightarrow \ldots \rightarrow A_z$ including a *MCS(Q)*, let a set of p-attributes in ρ be denoted by $A_p(\rho)$ where $MCS(Q) \subseteq A_p(\rho) \subseteq A_p(Q)$. Let T_τ^{tot} denote the total number of tuples generated in a fixed period τ and $T_\tau^{unm}(\rho)$ denote the number of unmatched tuples by the sequence ρ during the same period. The tuple dropping ratio $d_\tau(\rho)$ of the sequence ρ is defined as follows:

$$d_\tau(\rho) = \frac{T_\tau^{unm}(\rho)}{T_\tau^{tot}}. \quad (3)$$

If two or more candidate sequences have the same dropping ratio, the one whose last *ASC* has the lowest selectivity is chosen. The selectivity of an individual *ASC* is found when deciding the evaluation sequence of length 1 in the first phase. This procedure is continued repeatedly until the evaluation sequence contains all the p-attributes in $A_p(Q)$.

Example 4 In Fig. 3, let $\rho = A_1 \rightarrow A_2 \rightarrow A_3$ be a p-attributes sequence. If three tuples t_1, t_2 and t_3 are generated in a period τ, then the tuples t_1 and t_3 are dropped by the sequence ρ. Therefore, the tuple dropping ratio $d_\tau(\rho)$ is 2/3.

4.2.3 Implementation

In a monitoring module, for a fixed period τ, a partial macro-sequence is extended by one of the remaining p-attributes. Let n be the number of p-attributes. The time to find a new complete macro-sequence is $\tau^* (n-1)$ because all of the p-attributes should participate in the sequence. To measure the conditional selectivities of different candidate sequences in each period, an *instance counter* for each candidate sequence is needed. The instance counter of a candidate sequence keeps the cumulative summary of *1*'s in *GRB*'s resulting from evaluating all the incoming tuples on its candidate sequence for a specific period. Its value is utilized to compute the conditional selectivity of a p-attributes sequence in the first phase that finds *MCS(Q)*. On the other hand, in order to measure the tuple dropping ratios of different candidate sequences in each period, it is only checked whether an incoming tuple is dropped by a target candidate sequence or not. This task is performed by

examining the bit-values of *GRB* after evaluating the sequence. If they are all *0*'s, the tuple should be dropped.

Example 5 In Fig. 3, suppose that the tuples t_1, t_2 and t_3 are generated in the period τ_1, τ_2 and τ_3 respectively. Let ρ_i denote a macro-sequence whose length is i. For the tuple t_1 in the period τ_1, $\rho_1 = A_2$ because $s_{\tau 1}(A_1) = 3/4$, $s_{\tau 1}(A_2) = 1/2$, $s_{\tau 1}(A_3) = 3/4$ and $s_{\tau 1}(A_4) = 3/4$. Subsequently, for the tuple t_2 in the period τ_2, $\rho_2 = A_2 \rightarrow A_1$ because $s_{\tau 2}(A_2 \rightarrow A_1|A_2) = 0$, $s_{\tau 2}(A_2 \rightarrow A_3|A_2) = 1/2$ and $s_{\tau 2}(A_2 \rightarrow A_4|A_2) = 1$. At this point, since $ASC(A_2).qub \ \& \ ASC(A_1).qub = 0$, $MCS(Q) = \{A_2, A_1\}$. After finding *MCS(Q)*, for the tuple t_3 in the period τ_3, two candidate sequences $d_{\tau 3}(A_2 \rightarrow A_1 \rightarrow A_3)$ and $d_{\tau 3}(A_2 \rightarrow A_1 \rightarrow A_4)$ are compared. Since $d_{\tau 3}(A_2 \rightarrow A_1 \rightarrow A_3) = 1$ and $d_{\tau 3}(A_2 \rightarrow A_1 \rightarrow A_4) = 0$, $\rho_3 = A_2 \rightarrow A_1 \rightarrow A_3$. Accordingly, a macro-sequence $\rho_4 = A_2 \rightarrow A_1 \rightarrow A_3 \rightarrow A_4$ is found.

The processing time for finding a macro-sequence mainly depends on the number of p-attributes because its main task is comparing the conditional selectivity or dropping ratio of the remaining p-attributes one another. In the first phase (period) of a monitoring module, n p-attributes are compared one another so that one of the p-attributes is initially assigned to the macro-sequence. Subsequently, in the second phase, $(n-1)$ remaining p-attributes are compared. In the same way, the remaining p-attributes are compared until all the p-attributes are assigned to the macro-sequence. Therefore, the time complexity of finding a complete macro-sequence for $(n-1)$ periods is as follows:

$$n + (n-1) + (n-2) + \ldots + 1 = \sum_{i=0}^{n-1} (n-i) = 0\left(n^2\right). \tag{4}$$

4.3 Hybrid-sequence

Each region of an *ASC* can have a number of micro-arrows. A hybrid-sequence utilizes the filtering capability of a micro-arrow to enhance the performance of a macro-sequence. A procedure for finding a hybrid-sequence is similar to that for finding a macro-sequence. Starting from the first *ASC* of a macro-sequence, the possibility of employing any micro-arrow is examined. Given a partially identified hybrid-sequence $\rho' = \rho \rightarrow ASC(A_i)$ for a query set Q, if the region *ra[r]* of the $ASC(A_i)$ is visited by incoming tuples in a monitoring period τ, one micro-arrow is chosen among the candidate micro-arrows of the region $ASC(A_i).ra[r]$, based on their monitored filtering capabilities. The candidate micro-arrows are indicated in the candidate-arrow bitmap $ASC(A_i).ra[r].cab$. For the $ASC(A_j)$ satisfying that $A_j \notin A_p(\rho')$ and $ASC(A_i).ra[r].cab[j] = 1$, the filtering capability of a candidate micro-arrow $ASC(A_i).ra[r] \rightarrow ASC(A_j)$ is measured as the conditional selectivity $s_\tau(\rho' \rightarrow ASC(A_j)|\rho')$ if $A_p(\rho') \subset MCS(Q)$. Otherwise, it is measured as the tuple dropping ratio $d_\tau(\rho' \rightarrow ASC(A_j))$. The conditional selectivity of each candidate micro-arrow is slightly different from Definition 4. It identifies a set of uncovered queries based on the values of the global-result bitmap *GRB* evaluated by the sequence ρ'. In other words, in order to measure the conditional selectivity, only the queries whose corresponding bits of *GRB* are 1's are targeted. From the region $ASC(A_i).ra[r]$, if any candidate micro-arrow does not exist or its filtering capability is lower than that of the macro-arrow of $ASC(A_i)$ in a macro-sequence, the micro arrow of the region $ASC(A_i).ra[r]$ is not established. There can be several hybrid-

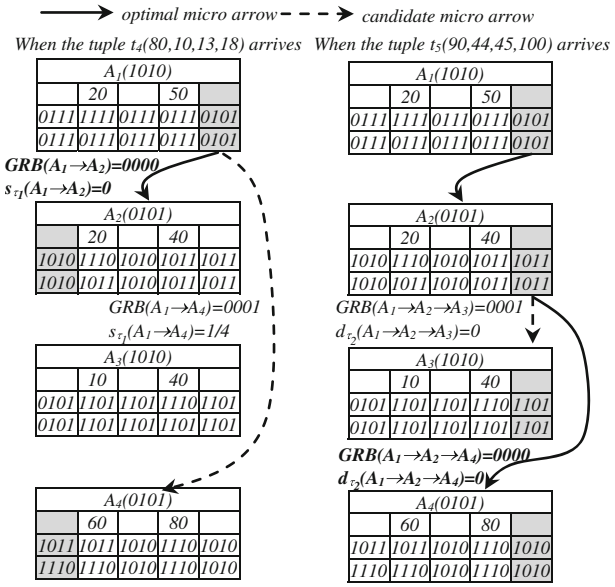

Fig. 4 The process of identifying a hybrid-sequence

sequences because each region of an *ASC* can possibly establish its own micro-arrow. These hybrid-sequences are expanded concurrently. The filtering capability of a specific hybrid-sequence is measured against only those tuples that have visited the series of the *ASC* regions in the sequence.

Example 6 In Fig. 4, suppose that the first *ASC* in a hybrid-sequence is $ASC(A_1)$ and the tuples t_4 and t_5 are generated in the period τ_1 and τ_2 respectively. For the tuple t_4 in the period τ_1, the micro-arrow becomes $ASC(A_1).ra[5] \rightarrow ASC(A_2)$ because $s_{\tau 1}(ASC(A_1).ra[5] \rightarrow ASC(A_2)) = 0$ and $s_{\tau 1}(ASC(A_1).ra[5] \rightarrow ASC(A_4)) = 1/4$. At this point, $MCS(Q) = \{A_1, A_2\}$ because $ASC(A_1).qub \ \& \ ASC(A_2).qub = 0$. Therefore, for the tuple t_5 in the period τ_2, $d_{\tau 2}(ASC(A_1).ra[5] \rightarrow ASC(A_2).ra[5] \rightarrow ASC(A_3))$ and $d_{\tau 2}(ASC(A_1).ra[5] \rightarrow ASC(A_2).ra[5] \rightarrow ASC(A_4))$ are compared. Since the former is 0 and the latter is 1, the next micro-arrow becomes $ASC(A_2).ra[5] \rightarrow ASC(A_4)$.

5 Evaluation and adaptation

5.1 Evaluation

If the run-time evaluation order is chosen to be a macro-sequence, its evaluation is performed according to the order of *ASC*'s in the sequence. In a hybrid-sequence, a macro arrow plays a role as the substitute of a micro arrow. The evaluation sequence firstly tries to employ the micro arrow that the visited region of the currently evaluated ASC owns. If it does not exist, the macro arrow on the macro-sequence is employed. The run-time evaluation order of their *ASC*'s may be different according to the p-attribute values of an incoming tuple. Due to this reason, a set of unevaluated

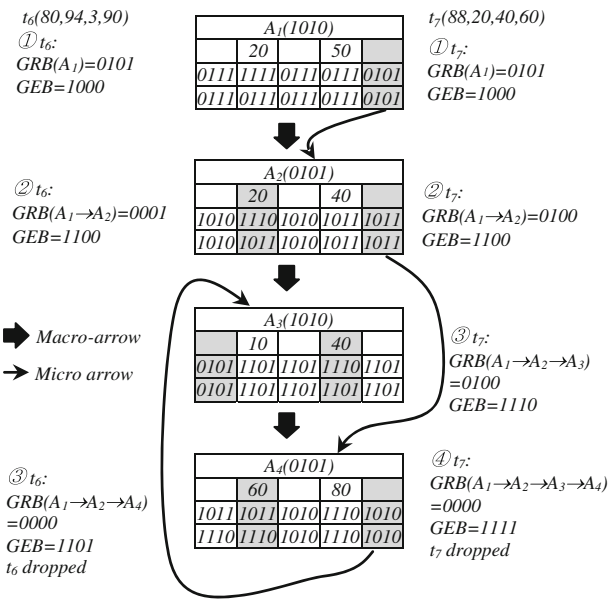

Fig. 5 Run-time evaluation by a hybrid-sequence

ASC's can be different, so that it should be traced continuously. For this purpose, an additional bitmap called a *Global Evaluation Bitmap (GEB)* is introduced. Given a set of p-attributes $A_p(Q)$, the bitmap *GEB* has $|A_p(Q)|$ bits each of which is corresponding to a distinct p-attribute. At first, it is initialized to all *0*'s. If the *ASC* of a specific p-attribute has been evaluated, the corresponding bit of *GEB* is set to *1*. Accordingly, the set of unevaluated *ASC*'s at a specific point can be identified by the bitmap *GEB*. Exceptionally, if there is no micro-arrow in the visited region of the evaluated *ASC*, among the unevaluated *ASC*'s, the most preceding *ASC* in the macro-sequence is evaluated.

Example 7 Fig. 5 illustrates how a hybrid-sequence is evaluated. Suppose that a macro-sequence is $\rho^{macro} = A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow A_4$ and there are three micro-arrows $\rho_1^{micro} = ASC(A_1).ra[5] \rightarrow ASC(A_2)$, $\rho_2^{micro} = ASC(A_2).ra[5] \rightarrow ASC(A_4)$ and $\rho_3^{micro} = ASC(A_4).ra[5] \rightarrow ASC(A_3)$. For the tuple t_6, according to its attribute values, the micro-arrows ρ_1^{micro} and ρ_2^{micro} are followed to process the tuple t_6 until it is dropped. For the tuple t_7, since there is no micro-arrow from the region $ASC(A_2).ra[2]$ containing $t_7[A_2] = 20$, the first unevaluated ASC $ASC(A_3)$ in the sequence ρ^{macro} is evaluated subsequently.

5.2 Adaptive rearrangement

Since the filtering capability of each p-attribute for a query set Q can vary dynamically as time goes by, the tuple dropping ratio of the current evaluation sequence of *ASC*'s should be monitored periodically in order to keep the sequence to be as efficient as possible. This period is called the *re-computation period* λ for a tuple dropping ratio. The *A-greedy* algorithm (Widom and Babu 2001) only addresses how to adaptively adjust the evaluation sequence of filters in a single query but it does

not specify exactly when to start it. In this paper, the current evaluation sequence is rearranged whenever its tuple dropping ratio is changed more than a specified threshold μ, called a *rearrangement threshold*. Given the current evaluation sequence ρ, let $d_{init}(\rho)$ denote the initial dropping ratio of the sequence ρ when it was selected to be the current sequence. On the other hand, let $d_{cur}(\rho)$ denote the currently monitored dropping ratio of the sequence ρ. Its value is computed against those tuples that are generated in the last re-computation period λ in order to reflect the recent variation of the dropping ratio. Whenever the following condition is satisfied, the arrangement of the *ASC*'s in the monitoring module described in Sections 4.2 and 4.3 is performed again to replace the current evaluation sequence.

$$\Delta d(\rho) = \left| \frac{d_{cur}(\rho) - d_{init}(\rho)}{d_{init}(\rho)} \right| \geq \mu \tag{5}$$

The proposed rearrangement scheme establishes the evaluation sequence for the future data elements of an underlying data stream based on the most recently passed data elements. As the value of μ is set to be smaller, the current evaluation sequence is more frequently rearranged but the run-time overhead is increased as well. Therefore, the value can control how precisely the current change of the data stream is reflected to the current evaluation sequence.

6 Experimental results

In this section, the performance of the proposed method is comparatively analyzed. All the algorithms are implemented in C, and all the experiments are executed on a Pentium 4 CPU 2.66 GHz system with 1G RAM. The system runs Linux with 2.4.5 kernel and gcc 3.3.2. For the following experiments, two different synthetic datasets and one real dataset are used to verify the effectiveness of the *ASC*'s arrangement strategy. Each of the synthetic datasets consists of *500,000* tuples and *20* integer-type attributes. The integer value of each attribute is generated from a range *[0..99]* but the data distributions of the two synthetic datasets D_1 and D_2 are different. While the former is uniform distribution, the latter is non-uniform distribution. For the real dataset D_3, a million US Census 1999 (http://kdd.ics.uci.edu, UCI KDD Archive) dataset is used. It has *10* integer-type attributes and *1,000,000* tuples. Furthermore, a number of different query sets are employed. The characteristics of the query sets are specified in Table 1. In this table, the item "*Standard deviation*" indicates the standard deviation for the number of *1*'s in the query-result bitmap of each region in an *ASC*. The overall evaluation cost *cost(Q, D)* of a query set Q for a dataset D is measured by the number of evaluated *ASC*'s, assuming that the cost of evaluating each *ASC* is identical.

Table 1 Specifications of experimental query sets

	Q_1	Q_2	Q_3					Q_4
Number of queries in Q	50	50	30	30	35	40	50	50
Number of selection predicates	204	173	78	81	98	138	173	362
Number of p-attributes	10	15	6	7	8	10	15	10
Standard deviation	0.08	0.2	0.06	0.06	0.15	0.1	0.25	varying

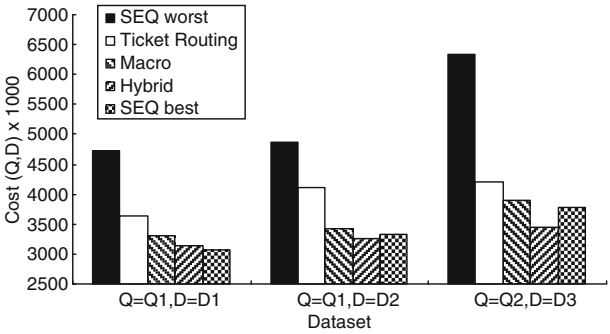

Fig. 6 Processing costs according to the evaluation sequences of *ASC*'s ($\mu = 0$)

In Fig. 6, the processing costs of various evaluation sequences for the *ASC*'s are compared along with the evaluation cost of *Ticket Routing* (Avnur and Hellerstein 2000). The cost of *Ticket Routing* is measured by the number of visited *selection modules*. The query set Q_1 is evaluated for the synthetic datasets D_1 and D_2, whereas the query set Q_2 is used for the real dataset D_3. The term *Macro* denotes the evaluation order of *ASC*'s by a macro-sequence. On the other hand, the term *Hybrid* denotes that by a hybrid-sequence. In these two schemes, the current evaluation sequence is rearranged adaptively. Assuming that the tuples of a target dataset arrive at a constant rate, the re-computation period λ for a tuple dropping ratio is set to every *10,000* tuples. Furthermore, the rearrangement threshold μ is set to *0*. It means that a rearrangement process for the current evaluation sequence is invoked whenever the period λ is elapsed. The terms *SEQ worst* and *SEQ best* denote the highest and lowest evaluation costs for a target dataset. These two costs are found by performing all the possible evaluation sequences of the *ASC*'s experimentally. Their evaluation sequences are not changed adaptively but fixed. As shown in Fig. 6, both *Macro* and *Hybrid* not only operate more efficiently than *Ticket Routing* but also approximate *SEQ best* in all the target datasets. Especially, in the dataset D_2 and D_3, *Hybrid* performs better than *SEQ best*. It means that the adaptive change of the current evaluation sequence for incoming tuples is more effective than the best fixed evaluation sequence. Also, it shows that micro-arrows can reduce the run-time evaluation cost.

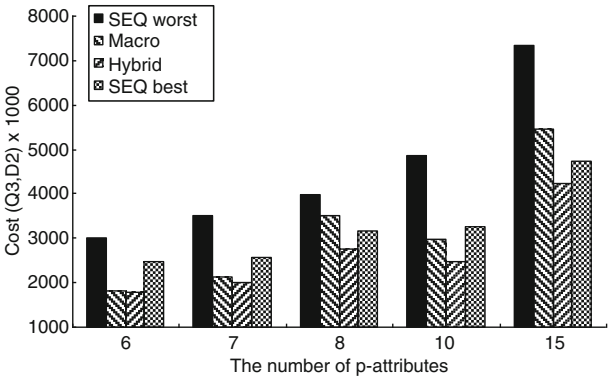

Fig. 7 Processing costs according to the number of pattributes ($\mu = 0$)

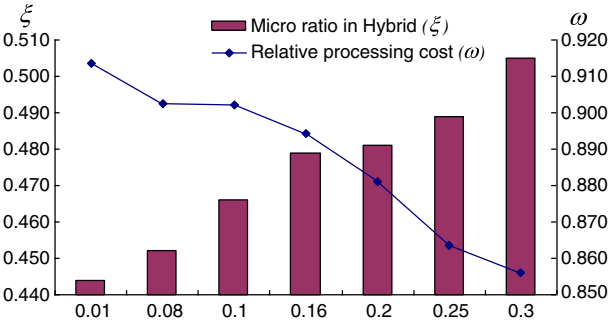

Fig. 8 The effectiveness of micro-arrows

Figure 7 shows the change of the processing cost of each scheme presented in Fig. 6 by varying the number of p-attributes. In this experiment, the query set Q_3 is processed for the dataset D_2. As in the experiment of Fig. 6, the value of λ in *Macro* and *Hybrid* is set to every *10,000* tuples. As the number of p-attributes is increased, the performance gaps among the four schemes are also comparatively enlarged. In addition, *Hybrid* performs better than *SEQ best*.

Figure 8 verifies the effectiveness of micro-arrows by showing that the overall query performance is propositional to the ratio of micro-arrows in an evaluation sequence. A term *micro ratio* ξ is defined as the ratio of the number of evaluated micro-arrows over the total number of evaluated macro/micro arrows. The dataset D_1 and the query set Q_4 are used for this experiment. In this figure, the *x*-axis indicates the standard deviation of Q_4 in Table 1. As the value gets larger, the gaps among the filtering capabilities of regions in an *ASC* are enlarged. The ratio of the processing cost of the *Hybrid* scheme over that of the *Macro* scheme is defined as a term *relative processing cost* ω. As the standard deviation is increased, the micro ratio is increased while the relative processing cost is decreased. Consequently, it leads to the improvement of query performance.

Figure 9 shows the effect of the adaptive rearrangement of the current evaluation sequence for the different values of the rearrangement threshold μ. To simulate the dynamic change of the selectivities of selection predicates, each of the datasets D_1 and D_2 is iteratively repeated as a sub-dataset to build a target dataset D. The value of λ is set to every 80, 000 tuples. The processing cost *cost*(Q_1, D) for the query set Q_1 is traced. Initially, the evaluation sequence for the first sub-dataset D_1 is used. As shown in this figure, whenever the boundary of a sub-dataset is crossed over, the

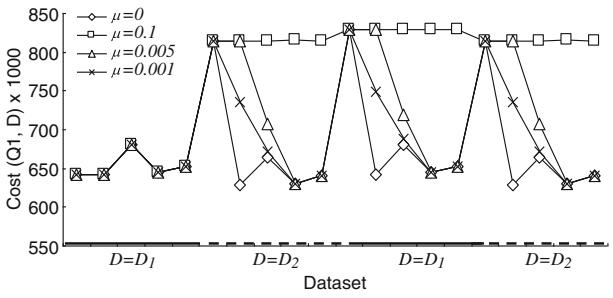

Fig. 9 The effect of adaptive rearrangement according to a rearrangement threshold μ

processing cost is rapidly increased, so that the current evaluation sequence is no longer optimal. Subsequently, a newly adjusted evaluation sequence is obtained by the adaptive rearrangement process as described in Section 5.2, which makes the processing cost be close to the optimal cost. In addition, as the value of μ is set to be smaller, the rearrangement process is invoked more frequently. Accordingly, the proposed method can approximate the optimal processing cost more rapidly. However, it also increases the run-time rearrangement overhead due to the frequent invocation of the rearrangement process.

7 Conclusions

In order to process the selection predicates of multiple continuous queries efficiently in a data stream environment, an attribute-based construct *ASC* and its matching algorithm are proposed in this paper. The proposed approach saves space usage by sharing the common selection predicates of target multiple continuous queries and also reduces run-time overhead by utilizing the pre-computed matching results of the queries based on the comparison constants of p-attributes expressed in the selection predicates of the queries at compile-time. In addition, the query performance is optimized by arranging the evaluation order of multiple *ASC*'s at two different levels and rearranging the current evaluation sequence adaptively. The goal of this optimization is minimizing the evaluation of unnecessary operations by dropping unmatched tuples as early as possible. A micro-arrow used in a hybrid-sequence can play an important role to achieve the goal further. The proposed method and optimization techniques are verified through various experiments.

Acknowledgements This work was supported by core research program (No. 2011-0016648) and NRL Program (No. 2010-0008007) of the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST).

References

Abadi, D., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., et al. (2003). Aurora: A new model and architecture for data stream management. *VLDB Journal, 12*, 120–139.

Avnur, R., & Hellerstein, J. (2000). Eddies: Continuously adaptive query processing. In *Proceedings of ACM SIGMOD international conference on management of data* (pp. 261–272). Dallas, TX.

Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of ACM symposium on principles of database systems* (pp. 1–16).

Babu, S., Motwani, R., Munagala, K., Nishizawa, I., & Widom, J. (2004). Adaptive ordering of pipelined stream filters. In *Proceedings of ACM SIGMOD international conference on management of data* (pp. 407–418). Paris, France.

Bizarro, P., Babu, S., DeWitt, D., & Widom, J. (2005). Content-based routing: Different plans for different data. In *Proceedings of the 31st international conference on very large data base* (pp. 757–768). Trondheim, Norway.

Chandrasekaran, S., & Franklin, M. J. (2002). Streaming queries over streaming data. In *Proceedings of the 28th international conference on very large dataBases* (August).

Chen, J., DeWitt, D. J., Tian, F., & Wang, Y. (2000). NiagaraCQ: A scalable continuous query system for internet databases. In *Proceedings of ACM SIGMOD international conference on management of data* (pp. 379–390). Dallas, Texas, USA.

Chen, J., DeWitt, D. J., & Naughton, J. F. (2002). Design and evaluation of alternative selection placement strategies in optimizing continuous queries. In *Proceedings of the 18th international conference on data engineering* (pp. 345–356). San Jose, CA, USA.

Fabret, F., Jacobsen, H. A., Llirbat, F., Pereira, J., Ross, K. A., & Shasha, D. (2001). Filtering algorithms and implementation for very fast publish/subscribe systems. *Proceedings of ACM SIGMOD international conference on management of data*.

Hanson, E., Chaaboun, M., Kim, C.-H., & Wang, Y.-W. (1990). A predicate matching algorithm for database rule systems. In *Proceedings of ACM SIGMOD international conference on management of data* (pp. 271–280).

Lee, H.-H., & Lee, W.-S. (2008). Attribute-based evaluation of multiple continuous queries for filtering incoming tuples of a data stream. *Information Sciences, 178*, 2416–2432.

Madden, S. R., Shah, M., Hellerstein, J. M., & Raman, V. (2002). Continuously adaptive continuous queries over streams. In *Proceedings of ACM SIGMOD international conference on management of data*. Madison, Wisconsin, USA.

Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., et al. (2003). Query processing, resource management, and approximation in a data stream management system. In *Proceedings of first Biennial conference on innovative data systems research* (pp. 245–256). Asilomar, CA.

Munagala, K., Srivastava, U., & Widom, J. (2006). Optimization of continuous queries with shared expensive filters. In *Proceedings of international conference on very large dataBase*. Seoul, Korea.

Sharaf, M. A., Chrysanthis, P. K., Labrinidis, A., & Pruhs, K. (2007). Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM Transaction on Database Systems, 33*(1), 1–42.

Wang, Y., Bai, S., Tan, J., & Guo, L. (2006). *Efficient Filtering Query Indexing in Data Stream. WISE 2006 Wokshops. LNCS*, 4256:1–12.

Widom, J., & Babu, S. (2001). Continuous queries over data streams. ACM SIGMOD Record: 109–120.

Wu, K.-L., Chen, S.-K., & Yu, P. S. (2006). Query indexing with containment-encoded intervals for efficient stream processing. *Knowledge and Information Systems, 9*(1), 62–90.