High Performance Stream Query Processing With Correlation-Aware Partitioning [*]

Lei Cao and Elke A. Rundensteiner
Worcester Polytechnic Institute, Worcester, MA USA
(lcao,rundenst)@cs.wpi.edu

ABSTRACT

State-of-the-art optimizers produce one single optimal query plan for all stream data, in spite of such a singleton plan typically being sub-optimal or even poor for highly correlated data. Recently a new stream processing paradigm, called multi-route approach, has emerged as a promising approach for tackling this problem. Multi-route first divides data streams into several partitions and then creates a separate query plan for each combination of partitions. Unfortunately current approaches suffer from severe shortcomings, in particular, the lack of an effective partitioning strategy and the prohibitive query optimization expense. In this work we propose the first practical multi-route optimizer named *correlation-aware multi-route stream query optimizer* (or CMR) that solves both problems. By exploiting both intra- and inter-stream correlations of streams, CMR produces effective partitions without having to undertake repeated expensive query plan generation. The produced partitions not only are best served by distinct optimal query plans, but also leverage the partition-driven pruning opportunity. Experimental results with both synthetic and real life stream data confirm that CMR outperforms the state-of-the-art solutions up to an order of magnitude in both the query optimization time and the run-time execution performance.

1. INTRODUCTION

Motivation. Given a user query, either static or continuous, most modern query optimizers determine a *single optimal* query plan with the lowest execution cost compared to other possible plans. This is based on the implicit assumption that stream data follows a uniform distribution. However, real-world data is often skewed with possibly complex correlations hidden in both a single stream or across multiple streams [16]. Thus using overall stream statistics as the base for constructing one single query plan serving all data tuples in the stream misses important opportunities for effective query optimization [14, 13].

Given skewed data, the recently emerged multi-route paradigm instead proposes that the input stream data should be divided into partitions such that each partition exhibits distinct statistical characteristics. This then leads us to support *multiple query plans* instead of one *single plan*, with each plan tuned to serve a combination of partitions, called a *partition query*. Let us demonstrate with a real-world example the optimization opportunities enabled by data correlations.

Motivating Example. Consider an application which monitors recent promising stocks by issuing the following query Q_m [13].

```
SELECT S.sector, S.company, S.price, C.pattern, N.sector
FROM Stock as S, News as N, CandlestickPatterns as C
WHERE matches(S.data[60 seconds], C)            /*op1*/
AND S.company = N.company[3 hours])             /*op2*/
WINDOW 60 seconds
```

The *CandlestickPatterns* table contains the most significant "Candlestick" patterns, e.g., "Engulfing" and "Kickers" [19] which are widely recognized as effective indicators of bullish stocks. Operator op_1 performs a similarity join on the latest financial data (i.e., incoming stock data from the last 60 seconds) with the *CandleStick-Patterns* table, while operator op_2 performs a match of the stock's name with the last three hours of Yahoo! Financial news (News). Here let δ_i denote the selectivity of op_i.

In the stock market the price of a company's stock is highly correlated with the health of the industry represented by the stock sector. Stocks in the same sector tend to exhibit similar trends. Due to these strong correlations between stocks performance and their sectors we may want to divide the stock stream into partitions by their sectors, such as Agriculture, Energy, Hi-tech, etc.

Suppose it is a bullish market. Based on the average selectivities with $\delta_1 > \delta_2$, the single query plan optimal for the overall data has the ordering $<op_2, op_1>$. Now suppose the news reports the Deepwater Horizon oil spill accident. This may hurt the performance of companies in the energy sector. Under this condition the energy sector stocks may have fewer matches with the *Candlestick-Patterns* table and more matches instead with the news. In this case, δ_1 will be relatively lower than δ_2 for such data. So $<op_1, op_2>$ becomes the most efficient processing order for this particular partition of data. Since other sectors are not influenced, the "global best join ordering" $<op_2, op_1>$ remains optimal for the other partitions. In this scenario, producing a customized query plan for the energy partition of data would clearly outperform a solution that forces the usage of one single generic plan upon all data types.

This horizontal partitioning of the stock stream exposes another promising opportunity for optimization when strong *inter-stream correlation* is experienced between stocks and news. The strong inter-stream correlation exists when the stock and news streams exhibit similar statistical properties. For example the media exposure of the companies may be also highly correlated to its industry sector. In this case it would be beneficial to also divide the news stream

[*]This work is supported by NSF grants: IIS-1018443 & 0917017.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.
Proceedings of the VLDB Endowment, Vol. 7, No. 4
Copyright 2013 VLDB Endowment 2150-8097/13/12.

by its sector attribute (as shown in Fig. 1).

We notice that after this coordinated partitioning across multiple streams the join operation among stocks and news no longer must be conducted among *all* pairs of partitions from these two streams. For instance in our example, assume each company only belongs to one single sector. The energy partition S_e of stock stream will only produce results when it joins with the energy partition N_e of the news stream (Fig. 1). Similarly S_a is guaranteed to not produce any output tuple when joining with partition N_a of news, because the domain values of the join attribute S.company in partition S_a do not overlap with the domain values of N.company in partition N_e. Clearly if in general we could successfully identify and prune all *join pairs* guaranteed to not produce any output tuples, a major performance benefit could be reaped. In short, the query performance can be radically improved by exposing and then exploiting inter-stream correlations between stock and news streams.

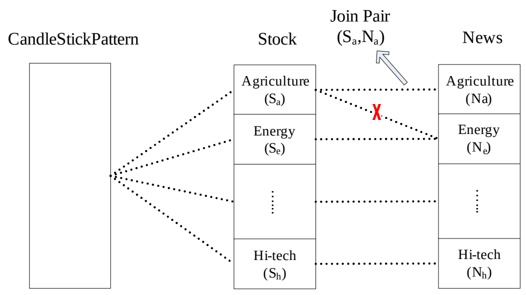


Figure 1: Join Pair Pruning for Query Q_m

Technical Challenges. Finding an effective multi-route solution which fully utilizes the optimization opportunities illustrated above is challenging. First as shown in [13] given a single stream *S* the complexity of dividing *S* tuples into partitions with distinct optimal execution plans is already exponential in the number of S's domain values even when only considering S's local data distribution. Worst yet to utilize the join pair pruning, the partitioning of one stream must also consider the partition decisions made for the other participating streams. However simply partitioning all streams on their join attributes may lead to not only too many partitions but also very poor ones, namely the tuples in such a partition may not share the same optimal plan.

To complicate matters further, with the introduction of partitioning query optimization now consists of two tasks: (1) determining the partitions of the input streams, and (2) creating a query plan for each partition query. Unfortunately the two subproblems are *strongly interdependent*. A change in one optimization decision requires re-optimization of the other subproblem. This yields a much larger optimization space than the one considered by traditional optimizers. This conflicts with the requirement that multi-route optimization must be highly efficient to be viable in fluctuating stream environments [17]. It is this challenging problem of the design of a practical multi-route optimizer meeting these competing demands that we tackle in this work.

State-of-the-Art. While initial work on multi-route processing has recently emerged [13, 20], the existing techniques do not address the above described challenges. To reduce the search space these multi-route optimizers construct partitions by either only considering the local statistics of each individual stream [13] in isolation or by going with the unrealistic simplification of assuming conditional independence across attributes [20]. By ignoring strong inter-stream correlations existing optimizers miss the critical join pair pruning opportunity illustrated above. To avoid enumerating the entire search space, they identify partitions either by employing off-the-shelf randomized partitioning algorithms [13] or by utilizing greedy search algorithms without backtracking [20]. These simplistic search heuristics tend to result in ineffective optimization decisions as confirmed by our experiments (Sec. 6.3). Furthermore in the search process they evaluate the quality of a selected partitioning by creating query plans for the corresponding partition queries. This tight interdependency of partitioning and query planning causes both methods to suffer from an order of magnitude increase of optimization time compared to a traditional optimizer.

Proposed Solution. In this paper we propose the first practical multi-route optimizer named *Correlation-Aware Multi-Route Stream Query Optimizer* (CMR). By explicitly exploring the correlations at both the intra- and inter-stream levels, CMR produces an effective multi-route solution within a short optimization time. CMR is built on two key principles established by characterizing the situational context in which a multi-route approach would be a preferred solution over the classical single plan approach.

First, a multi-route solution shows a significant performance advantage only when the streams can be divided into partitions with distinct statistics relative to other partitions, yet with uniformity among tuples within the same partition. This intra-stream correlation property is called *skewed uniformity*. Based on this property we design an algorithm called CORBA that discovers appropriate partitions of each stream based on the *collision probability theory* [7]. The sampling-based nature and the sub-linear sample complexity of CORBA make it ideal for unbounded continuously arriving streaming data with dynamically changing statistics.

Second, we demonstrate that strong inter-stream correlation referred in the joint distribution across input streams is the second critical prerequisite for the deployment of a multi-route solution being beneficial. In that case the multi-route approach will not only avoid the prohibitive overhead of simultaneously maintaining a large number of execution plans but also benefit from the highly effective join pair pruning optimization. We propose a *information theory*-based metric [3] that can determine the existence of strong inter-stream correlation. Better yet, this metric can now also be exploited to further optimize the initial partitioning by merging partitions with similar joint statistics.

The two *correlation indicators* not only assist us in deriving an effective partitioning solution without having to repeatedly undertake expensive query plan generation, but also offer an *early termination* mechanism. This assures that CMR quickly converges to the traditional "single plan" approach when the data streams show neither skewed uniformity nor strong inter-stream correlation.

Our experiments (Sec. 6.3) confirm that *CMR* outperforms state-of-the-art methods up to an order of magnitude in both its runtime execution performance and its optimization time. Thus CMR is a win-win; it not only produces a highly effective multi-route solution, but does so with a surprising low optimization expense.

Contributions. In summary, the contributions of this work include: 1) We introduce *correlation indicators* that reliably determine the applicability of the multi-route approach for a given workload. 2) We design a sample-based partitioning algorithm that discovers effective *intra-stream* partitions in sub-linear sample complexity. 3) We propose a partition optimization approach that based on the *inter-stream correlation indicator*, is shown to successfully exploit the join pair pruning opportunity. 4) Our experiments confirm that our *CMR* optimizer integrating the above strategies into one comprehensive solution framework outperforms the existing alternatives up to an order of magnitude.

2. RELATED WORK

Streaming Databases. Like Query Mesh [13], CMR is also a multi-route approach. Once an effective multi-route solution has been identified by CMR, it can be successfully executed by any multi-route infrastructure, including the Query Mesh executor. However, this is where key similarities end. Unlike our effort, Query Mesh [13] does not explore the data correlations within a single stream nor across streams for stream partitioning. Rather it uses a simple *randomized partitioning strategy* for every stream in isolation, thus potentially leading to ineffective partitioning. Furthermore Query Mesh relies on query planning to evaluate if a partitioning is good, making the optimization process prohibitively expensive. CMR overcomes both bottlenecks of Query Mesh, namely the ineffective partitioning and the expensive optimization.

The *content-based routing* (CBR) extension of Eddies [14] continuously profiles operators and identifies "classifier attributes" to partition the underlying data into tuple classes to be routed by Eddies. However CBR only considers the special case of a single-attribute decision. We take a more general approach in CMR by incorporating correlations of multiple attributes not only within one stream but also across several streams. CBR inherits several problems associated with Eddies, such as continuous and often unnecessary fine-grained route re-learning at runtime. Extending CBR to non-Eddies-based systems, i.e., systems that pre-compute plans prior to execution is non-trivial, as CBR does not compute full routes. In contrast, our CMR has a much wider scope of applicability as it addresses multi-route query processing in plan-based systems−the standard paradigm for data processing.

Static Databases. In the context of static relational databases, Polyzotis proposes an iterative algorithm [15] which partitions *one relation* of a join query into *k* (a fixed pre-defined threshold) partitions and constructs *k* join plans, one for each partition. However all other n - 1 relations are left non-partitioned. Our work does not impose these two artificial limitations. First, we allow the partitioning of multiple relations. Second, we do not apriori impose a rigid number k of partitions that a relation is forced to be divided into. Instead we use data-driven criteria for determining if or if not to partition and if so how many partitions to make.

In [20] Tzoumas et al. introduces the notion of *conditional join plans*, a restricted search space resulting from horizontal partitioning that captures both the partitioning and join ordering aspects. However [20] makes partitions only based on one single attribute, while our work can partition on any set of attributes. They estimate the join selectivity with the assumption that *conditional independencies* exist in the data. Instead by partitioning the data with the guidance of both the intra-stream and inter-stream correlations, we accurately estimate the selectivity without this assumption.

In [8] Herodotou et al. proposes techniques to generate efficient plans for SQL queries on already *pre-partitioned* tables. In other words they only focus on the traditional optimization problem, namely they produce an execution plan for each partition query rather than the harder problem of identifying a suitable partitioning within this huge partitioning search space. The latter is instead tackled by our work.

Parallel and Distributed Data Processing. Like multi-route optimization, data partitioning also has been considered in parallel and distributed settings [1, 6, 11]. In these works, streaming data is partitioned across machines of a parallel system and then queries are rewritten accordingly to use the fragments located on each node. However unlike multi-route optimization, the goal here is load balancing, namely to spread the load of a query across the distributed system so to avoid over-loaded nodes and reduce heavy network traffic. In other words the data partitioning does not aim to produce partitions with distinct optimal query plans. Instead they aim to have equal load on each machine. However producing a distinct plan for different partitions is the core concept of multi-route optimization, while making partitions equal in cost to keep a balanced workload across machines is not our concern.

3. MULTI-ROUTE OPTIMIZATION PROBLEM FORMULATION AND ANALYSIS

3.1 Problem Formulation

Problem Formulation. S_i is a data stream whose tuple values come from the universe $\mathcal{U}_i$ (1≤i≤m). A query *Q* is specified on a set of streams $\{S_1,S_2,...,S_m\}$. A *partition solution* of S_i, $P(S_i)$, is a set of pairwise disjoint subsets of tuple values e^{S_i} from $\mathcal{U}_i$ whose union covers the universe $\mathcal{U}_i$. A subset of stream S_i containing only the tuple values in e^{S_i} is called a substream of S_i. It is also denoted as e^{S_i} when no ambiguity arises.

We use $P(S_i)$ to denote a particular partition solution of stream S_i and $\mathfrak{P}(S_i)$ to denote the set of all possible partition solutions of stream S_i. A *global partition solution* GP_j of the query *Q* on *m* streams S_i (1≤i≤m) is a set of m partition solutions $\{P(S_1),P(S_2),...,P(S_m)\}$. Given a GP_j of Q, a *partition query* $Q_p(GP_j)$ of query *Q* is a query identical to Q in semantics, but applied to a set of *m* substreams $\{e^{S_1},\ e^{S_2}\ ,...\ e^{S_m}\}$ of the streams $\{S_1,\ S_2\ ,...\ S_m\}$, with e^{S_i} being an element of $P(S_i) \in GP_j$, 1≤i≤m.

Given the set of all partition queries with respect to GP_j, denoted by $\mathbb{Q}(GP_j)$, a *multi-route query plan* $QP(Q, GP_j)$ of query *Q* for GP_j is a set of query plans, namely one for each of the partition queries in $\mathbb{Q}(GP_j)$. Now we are ready to define the notion of an optimal multi-route plan of GP_j.

Definition 1. *The optimal multi-route plan of* GP_j *denoted as* $QP^{opt}(Q, GP_j)$ *is the multi-route plan* $QP(Q, GP_j)$ *that processes all partition queries of* GP_j *with the minimal overall query execution costs denoted as* $cost(Q, GP_j)$.

Our problem of *multi-route optimization* can now be formally defined as follows.

Definition 2. *Given m sets* $\mathfrak{P}(S_1)$, $\mathfrak{P}(S_2)$, ..., $\mathfrak{P}(S_m)$ *of partition solutions of streams* S_1,S_2,...,S_m*, the* ***multi-route optimization problem*** MR_{opt} *is to find a global partition solution* GP_k *and the multi-route plan* $QP^{opt}\ (Q, GP_k)$*, where* GP_k *is formed by selecting a* $P(S_i)$ *from each* $\mathfrak{P}(S_i)$*, such that* $cost(Q, GP_k)$ *is minimal among the* $cost(Q, GP_j)$ *of all possible* GP_j *of Q.*

The MR_{opt} problem has a much larger optimization space than the one considered by traditional optimizers as analyzed below.

3.2 Complexity Analysis of Multi-Route Optimization Problem

We analyze the complexity of the multi-route optimization problem for a query *Q* with *m* input streams. Let C_i=$|\mathcal{U}_i|$ denote the cardinality of the universe of stream S_i. Then the *Bell number* [10] B_i in Eq. 1 represents the cardinality of $\mathfrak{P}(S_i)$.

$$B_i = \sum_{k=1}^{n_i} \left(\frac{1}{k!} \sum_{j=1}^{k} (-1)^{k-j} \binom{n_i}{k} j^{n_i} \right) \qquad (1)$$

To find the optimal global partition solution GP_k, $\prod_{1 \le i \le m} B_i$ possible combinations of *m* partition solutions from *m* streams respectively (possible GP_j) would have to be examined. E denotes the time complexity of the join query planning algorithm for *Q* on a given set of streams to join. If the query planning is applied to each partition query in each possible GP_j, then the overall complexity of the problem search space would be *O*($\prod_{1 \le i \le m} B_i$ *$*E$*). In general the problem of identifying an optimal plan for a given join query is known to be *NP*-hard [17]. B_i being exponential in C_i further exasperates the problem [13]. Clearly for large C_i, the search is prohibitively expensive. In essence, the complexity of MR_{opt} originates from two key factors: (1) the combinatorial number of possible GP_j and (2) the strong interdependency of the partitioning and query planning. Given the exponential complexity, it is thus imperative that efficient yet effective search heuristics are devised for tackling the MR_{opt} problem defined in Def. 2.

3.3 The CMR Approach

Any viable multi-route technique has to simplify this MR_{opt} problem, while still achieving an effective solution.

Possible Solution Approaches. Several methods are possible to reduce the MR_{opt} problem. One solution is to partition each stream in isolation by only considering the local stream statistics. However even if locally optimal solutions could be found, simply combining the local optimal partitions of each stream is not likely to produce an effective global multi-streams partitioning GP_j due to the ignorance of the statistical properties of other streams.

An alternative solution may be to impose some artificial restrictions on the possible partitioning or the query plans to be considered such as always dividing each stream into a fixed number of partitions or selecting optimal query plans only from a subset of all possible plans. However after applying such drastic restrictions, many effective multi-route solutions would be missed.

Our CMR Approach. CMR proposes a *correlation-aware partitioning* strategy to simplify this MR_{opt} problem. By leveraging both the intra- and inter-stream correlations for partitioning, this strategy divides the problem of searching for an effective GP_j into two sub-problems, namely *intra-stream partitioning* and *inter-stream partitioning*. Furthermore it decomposes the *partitioning and query planning* into two separate stages. Therefore CMR significantly simplifies the MR_{opt} problem, rendering it practical.

CMR is composed of three layers, namely intra-stream partitioner, inter-stream partitioner, and partition query planner as shown in Fig. 2. CMR first solves the intra-stream partitioning problem, namely produces an initial partitioning for each individual stream based on local stream statistics. These local partitions are then exploited to model the joint distribution across multiple streams.

As second step, the inter-stream partitioner further optimizes the initial local partitions and produces the final GP_j. The inter-stream partitioner is based on our proposed partition abstraction level joint distribution. This significantly *reduces* the optimal GP_j search space. Otherwise it would be composed of combinatorial number of possible GP_j if the partitioning instead relies on the much more fine-grained tuple value granularity joint distribution.

Furthermore the inter-stream partitioning is driven by our proposed joint statistical property based metric. This is highly effective compared to having to test each tried GP_j by an expensive query planning process. The query planning is only applied as the third and last step to produce the corresponding $QP^{opt}\ (Q, GP_j)$ after the final GP_j has been formed by CMR optimizer. Therefore CMR successfully decomposes the partitioning and query planning.

Effectiveness of CMR. The effectiveness of CMR lies on the particular statistical properties that are proposed as *indicators* for the multi-route optimization paradigm to be a suitable design choice, namely the strong intra- and inter-stream correlations.

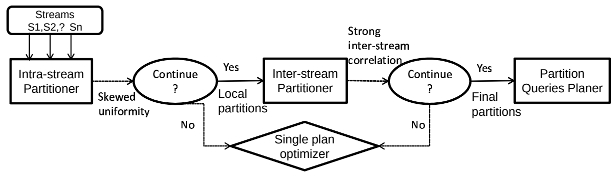


Figure 2: CMR Framework

First, our intra-stream partitioner produces an initial partitioning of each individual stream by discovering highly correlated tuples. This intra-stream partitioning serves as a good starting point of our inter-stream partitioner. That is, it naturally follows the intra-stream correlation that is indicator of a situation where the multi-route solution is beneficial.

Second, when a strong inter-stream correlation exists among the input streams, the tuples in the same stream partition formed by the intra-stream partitioner tend to also show similar joint distribution statistics. Thus modeling the joint distribution at the partition abstraction level would not miss the major characteristics of the joint distribution. We thus postulate that relying on the partition level joint distribution model to improve the local partitioning would not miss important optimization opportunities.

Third, the joint distribution captures both the data statistics of the input streams and the user query semantics. This leads to the following two observations. First, since the query optimizer uses the joint distribution to determine the optimal query plan, tuples sharing an optimal or a near optimal query plan could be located based on their joint distribution statistics. Second, the potential performance of each GP_j can be represented by some joint distribution statistics related metric. This leads to the important insight that it is possible to identify an effective GP_j by maximizing the metric instead of employing the expensive query planning process to explicitly evaluate the cost of each of the exponentially many GP_j.

As shown in Sec. 5, our inter-stream partitioner incorporating the above observations finds an effective GP_j by maximizing our proposed *mutual information* metric [3]. Our empirical study shows that this metric is a good indicator of the potential gain of a GP_j.

In summary in this work we propose the correlation-aware multi-route stream query optimizer (or CMR) to tackle this challenging MR_{opt} problem (Def. 2). By leveraging the two proposed *correlation indicators*, CMR successfully reduces the search space of the MR_{opt} problem. Yet it still guarantees to produce an effective partitioning which can take full advantage of the optimization opportunities afforded by the multi-route query processing paradigm.

4. CORRELATION-AWARE INTRA-STREAM PARTITIONING

Next, we will establish the fundamental insight that for a multi-route solution to be an effective solution for processing a query *Q*, each input stream should satisfy a certain statistical property. This observation opens the opportunity to effectively partition each single stream by modeling it as a uniform interval detection problem. Fortunately, we demonstrate that uniform intervals can be quickly discovered based on the theory that any interval close to being uniform would have a small *pairwise collision probability* [7]. This probability can be measured on a stream sample with sub-linear sampling complexity, making *CORBA* efficient and thus conductive to the dynamic streaming context. CORBA discovers appropriate partitions that build a solid foundation to render the inter-stream partitioning problem introduced in Sec. 5 practical.

4.1 The Skewed Uniformity Property

Multi-route optimization is based on the insight that tuples with similar statistical properties are likely to be best served by the same route [14]. On the one extreme if the data is uniform, then the traditional "*one single plan*" solution suffices to serve most of the tuples well. On the other extreme if the data is "too skewed", e.g., most tuples have distinct statistics, route-less solutions like Eddies [2] maybe most effective. Such route-less solutions at runtime decide for every tuple which operator to visit next. Thus they come at the cost of a per-tuple routing overhead. As illustrated in the motivation example in Sec. 1, a multi-route solution exhibits a major performance advantage in the case when each stream can be divided into partitions with distinct statistics, yet with uniformity among tuples within the same partition.

Here we formally define the above local stream property also being referred to as *skewed uniformity*. Let S_i be a data stream with a universe $\mathcal{U}_i = \{u_1, u_2, ...u_L\}$. By mapping $\mathcal{U}_i$ to a numerical universe $\mathcal{U}_L = \{1...L\}$, the *probability distribution* of S_i is represented as a function D: $\mathcal{U}_L \to [0,1]$. For each element $e_i \in \mathcal{U}_L$, *D(e_i)* measures the probability that e_i appears in stream S_i denoted as p_i. Given any interval $I \subseteq \mathcal{U}_L$, the probability of interval I is denoted as $p_I = \sum_{e_i \in I} p_i$.

Definition 3. *Given a distribution D: $\mathcal{U}_L \to [0,1]$, a partition threshold τ ($\tau \ll L$), and an error threshold ϵ, D is a distribution with* ***skewed uniformity*** *if:*

(1) There exists a distribution $\hat{D}$: $\mathcal{U}_L \to [0,1]$ which is represented as a sequence of disjoint intervals I_j and a corresponding sequence of values v_j ($1 \le j \le k$ with $k \le \tau$), where $v_j \le 1$; and

(2) Each interval I_j satisfies the ***uniform interval criteria****: $\sum_{i \in I_j}(p_i - v_j)^2 \le \epsilon^2 p_{I_j}$ [7]; and*

(3) $\|D - \hat{D}\|_2 = \sqrt{\sum_{I_j \in D}\sum_{i \in I_j}(p_i - v_j)^2)} \le \epsilon$.

*$\hat{D}$ is said to be an ϵ-**approximation** of D.*

Based on the above observation we propose to build an intra-stream partitioning for a given stream S_i with skewed uniformity by identifying the ϵ-approximation of D. This intra-stream partitioning serves as a good start point of our inter-stream partitioner. That is, it naturally follows the statistical property that is indicator of a situation where the multi-route solution is beneficial.

4.2 CORBA Algorithm

Capturing the distribution properties of streaming data is challenging due to the unbounded nature of the continuously arriving streaming data and its frequently changing statistics. Any approach involving prohibitive computational costs or producing results only after seeing the complete data stream is clearly not practical.

We now propose a collision probability-based algorithm named *CORBA* to partition each input stream into uniform intervals. Given a distribution D, *CORBA* will detect whether the skewed uniformity holds. If it holds, CORBA also outputs a partitioning of D with no more than τ partitions, namely the *ϵ-approximation* $\hat{D}$ (Def. 3).

The general idea of *CORBA* is to partition $\mathcal{U}_L$ into k ($k \le \tau$) longest intervals with each being approximately uniform, so called *longest uniform interval*. An interval $I[start, end]$ is said to be a longest uniform interval if there does not exist any other uniform interval $I' = [start', end']$ with $start' \le start$ and $end' \ge end$.

Let us assume that we have at our disposal an algorithm *detectLongUI* which given a point *low* in $\mathcal{U}_L$, can detect the uniform interval's boundary starting from *low* in *D*. Then the CORBA algorithm will perform as described in Alg. 1. At first $\hat{D}$ is initialized to be empty (Line 2). In each iteration, the algorithm searches the

Algorithm 1 CORBA(D, τ)

```
Input: distribution D, partition number threshold $\tau$
Output: Bool skewUniformity, partitioning $\hat{D}$ with no more than $\tau$ intervals. Each
    interval is represented as interval(low,high).
1:  low = 1, boundary = 1;
2:  $\hat{D}$ = $\phi$;
3:  skewUniformity = false;
4:  for $i = 1$ to $\tau$ do
5:    boundary = detectLongUI(D, low);
6:    $\hat{D}$ = $\hat{D}$ + interval(low,boundary);
7:    if (boundary == L) then
8:      skewUniformity = true;
9:      return;
10:   else
11:     low = boundary + 1;
12:   end if
13: end for
```

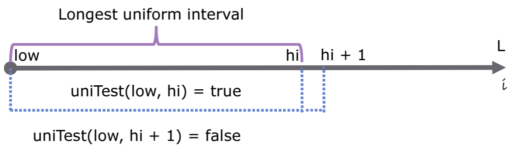


Figure 3: Longest Uniform Interval

interval boundary by running the *detectLongUI* algorithm (Line 5). In Line 6, $\hat{D}$ is updated by adding the newly identified *interval* to the set of intervals so far. If all elements of *D* have been covered by the located intervals, then the "for loop" (Line 4) will terminate early even if *the number of the intervals* has not reached the upper bound τ. In that case distribution D is said to be a distribution with *skewed uniformity*. If not, we start to search for a new interval with its start point set as the next element of the previously located boundary (Line 11). If after τ iterations, some elements of D remain uncovered, then the distribution D will be declared as not meeting the skewed uniformity property.

Above assumes the ability to detect the longest interval, for which we now introduce the **detectLongUI** algorithm. It is based on the **monotonic property** of a uniform distribution.

LEMMA 1. ***Monotonic Property****: If a given interval I[x,y] of distribution D does not correspond to a uniform distribution, then neither will its superset $I'[x', y']$ ($x' \le$ x and $y' \ge$ y). Naturally if I[x,y] is uniform all its subsets are also uniform.*

This lemma proven in [7] leads to the following observation.

Observation 1. *If an algorithm* ***uniTest*** *exists which, given an interval I = [x, y] $\subseteq \mathcal{U}_L$, could determine whether I is uniform, then given a point low $\in \mathcal{U}_L$ the boundary of the longest uniform interval starting from low can be identified by searching for a point hi $\in \mathcal{U}_L$ which satisfies the following conditions: uniTest(I[low, hi]) returns true, while uniTest(I[low,hi+1]) returns false.*

As shown in Fig. 3 this observation provides a criteria to determine the boundary of a uniform interval starting from point low, namely the last point hi that satisfies the uniTest.

Based on Lemma 1 and Obs. 1, the problem of detecting the boundary of a longest uniform interval can be solved using a binary search style algorithm. As shown in Fig. 4, given a start point *low* and end point *hi*, the interval $[low, mid]$ is tested first ($mid = low + \ulcorner \frac{hi-low}{2} \urcorner$). If uniTest returns true, then the possible range of the boundary shrinks to $[mid, hi]$. Otherwise it is reduced to $[low, mid - 1]$. The search procedure continues until only one candidate is left for the possible range value of the boundary.

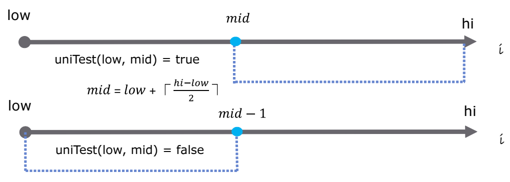

low
mid
hi
uniTest(low, mid) = true
$mid = low + \lceil \frac{hi-low}{2} \rceil$
low
$mid-1$
hi
uniTest(low, mid) = false

Figure 4: detectLongUI algorithm

uniTest Algorithm. Next we introduce our sampling-based uniformity test approach *uniTest*. This *uniTest* algorithm first proposed in [9], is based on the notion of *pairwise collision probability* explained next. Given a set of samples SA of *S*, let SA_I represent the set of samples that fall into I. The *pairwise collision probability* of *I* is the probability of getting the same tuple when randomly picking two samples from SA_I. This can be calculated as below.

Definition 4. *The* ***pairwise collision probability*** *of interval I is denoted by* $CP_I = \frac{collision(SA_I)}{\binom{|SA_I|}{2}}$ *where* $collision(SA_I) = \sum_{e_i \in I} \binom{|occur(e_i, SA_I)|}{2}$*, while* $occur(e_i, SA_I)$ *corresponds to the number of occurrences of* e_i *in* SA_I*.*

Intuitively CP_I will be small when *I* is close to being uniform and large if *I* is skewed. For example, in the extreme case if p_i of all elements e_i in *I* is 0 except for one element e_j, CP_I will be 1, since any two tries will be guaranteed to get the same tuple. This property of CP_I is similar to the l_2 norm of p_I, denoted by $\|p_I\|_2^2 = \sum_{i \in I}(\frac{p_i}{p_I})^2$. The quantitative connection between these two concepts can be established with the following lemma [9].

LEMMA 2. *If more than m* = $\frac{64}{\epsilon^4}$ *samples are collected from S with* ϵ *as the threshold in Def. 3, we have: (1)* $p[\,|CP_I - \|p_I\|_2^2| \le \frac{\epsilon^2}{2\hat{p}_I}] > \frac{3}{4}$*, (2)* $\hat{p}_I \ge p_I$*, where* $\hat{p}_I = \frac{2|SA_I|}{m}$ *and* CP_I *as Def. 4.*

Lemma 2 indicates that the *collision probability* of interval *I* is close to the l_2 norm of the real distribution of *I* when enough samples are collected. Since distributions close to being uniform have a small l_2 norm [7], we can conclude that given a random sampling of the stream *S*, any $I \subseteq \mathcal{U}_L$ with a small collision probability with high likelihood corresponds to a uniform distribution. Based on this conclusion, the *uniTest* algorithm that identifies the uniformity of interval *I* can now be devised as shown in Alg. 2.

Algorithm 2 uniTest(I[low,high], ϵ)

```
Input: portion I with its boundaries, error threshold $\epsilon$;
Output: Bool uniform
1: collect n = $16\ln(6L^2)$ groups of samples from $S$ denoted as $SA^1, \ldots, SA^n$,
   each with size m = $\frac{64}{\epsilon^4}$
2: for $i = 1$ to n do
3:     $\hat{p}^i(I) = \frac{2|SA_I{}^i|}{m}$;
4: end for
5: uniform = false;
6: if $Z_I$ = median$(CP_I{}^1, \ldots, CP_I{}^n) \le \frac{1}{|I|} + max_i\{\frac{\epsilon^2}{2\hat{p}^i(I)}\}$ then
7:     uniform = true;
8: end if
9: return uniform;
```

The uniTest algorithm is proven to be able to correctly evaluate the uniformity of a given interval *I*. More specifically given a interval *I* if uniTest returns true, then:

$$\sum_{i \in I}(p_i - \frac{p_I}{|I|})^2 \le \epsilon^2 p_I \qquad (2)$$

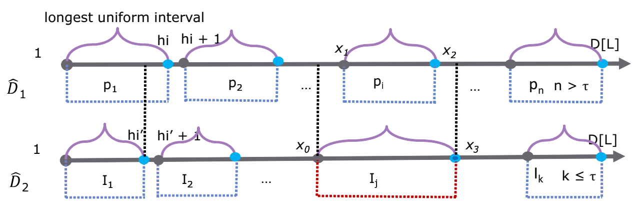


Figure 5: Contradictory Example

That is, I is uniform by Def. 3.

Complexity Analysis of uniTest. We first analyze the complexity of uniTest. As shown in Line 1 of Alg. 2, the sample complexity of *uniTest* is $O(\ln L^2 \epsilon^{-4})$. This is sub-linear in the domain size L. The run time complexity of *uniTest* is determined by the cost of calculating the *collision probability* of each domain value (Line 3). It is linear in L. Line 3 is looped $16\ln 6L^2$ times. Therefore the overall execution time of *uniTest* is $O(L\ln L^2)$.

Complexity Analysis of CORBA. We now are ready to analyze the complexity of CORBA. The sample complexity of *CORBA* is determined by the sample complexity of *uniTest* which is $O(\ln L^2 \epsilon^{-4})$ as analyzed above. Then the run time complexity of *CORBA* (Alg. 1) can be analyzed as follows. The body of the for-loop at Line 4 of (Alg. 1) executes at most τ times. Within this for-loop the computation in Lines 6 to 12 takes constant time. At Line 5 the time complexity of the algorithm *detectLongUI* is $O(L\ln L^3)$ (*detectLongUI* calls the *uniTest* algorithm $O(\ln L)$ times with the running time complexity of *uniTest* being $O(L\ln L^2)$). Hence the total execution time of *CORBA* is $O(\tau L \ln L^3)$.

4.3 Correctness of CORBA Algorithm

In this section we show the effectiveness of the *CORBA* algorithm by proving the following lemma.

LEMMA 3. *Given a distribution D, an interval number threshold* τ*, an error threshold* ϵ*, then (D is a skewed uniform distribution)* $\iff$ *(CORBA returns true).*

Proof: Proof in two directions.

"$\Rightarrow$." First we prove if *CORBA* accepts D, D will be a skewed uniform distribution. Assume *CORBA* produces an ϵ-*approximation* $\hat{D}$ of D with a sequence of uniform intervals $I_1, \ldots I_k (k \le \tau)$ and a corresponding sequence of values $\frac{p_{I_1}}{|I_1|}, \ldots, \frac{p_{I_k}}{|I_k|}$. By Eq. 2, the l_2 distance $dist_D$ between D and $\hat{D}$ is calculated as:

$$dist_D = \sum_{I_j \in D}\sum_{i \in I_j}(p_i - \frac{p_{I_j}}{|I_j|})^2 \le \epsilon^2 \sum_{I_j \in D} p_{I_j} = \epsilon^2 \qquad (3)$$

Therefore *D* is a skewed uniform distribution by Def. 3.

"$\Leftarrow$." Next we prove by **contradiction** if CORBA rejects D, then D is guaranteed not to be a skewed uniform distribution.

Given a distribution $\hat{D}_1$ which covers $\mathcal{U}_L$ with n longest uniform intervals ($n > \tau$). Suppose there exists another distribution $\hat{D}_2$ such that D satisfies Def. 3 as shown in Fig. 5. Since P_1 in $\hat{D}_1$ is the longest uniform interval starting from element e_1 of $\mathcal{U}_L$, the size of I_1 in $\hat{D}_1$ denoted by $|I_1|$ must be shorter than $|P_1|$. Therefore $hi' < hi$ as shown in Fig. 5. However in order to cover $\mathcal{U}_L$ with less intervals than $\hat{D}_1$, there must be at least one interval $I_j(x_0, x_3)$ in $\hat{D}_2$ which is the superset of some other interval $P_i(x_1, x_2)$ in $\hat{D}_1$ (see Fig. 5). This contradicts the assumption that P_i is a longest uniform interval. Therefore $\hat{D}_2$ cannot cover $\mathcal{U}_L$ with less intervals than $\hat{D}_1$. There does not exist any ϵ-*approximation* of D. Thus D cannot be a skewed uniform distribution. ■

5. CORRELATION-AWARE INTER-STREAM PARTITIONING

Given as input each stream broken into non-overlapping partitions by CORBA, we first establish a joint distribution property, namely strong inter-stream correlation across input streams. We then take an information theory based approach to measure this property. That is, we map each stream to a random variable and measure the inter-stream correlation with the *mutual information* metric [3]. This metric is empirically shown to be a good indicator of the potential gain of a multi-route solution. Guided by this foundation, we then propose a solution to further optimize the partitioning produced by CORBA. Our solution, called <u>CO</u>rrelatio<u>N</u> metri<u>C</u> g<u>U</u>ided pa<u>R</u>titioning or *CONCUR*, produces an effective global partition solution by merging partitions so to maximize the mutual information metric.

5.1 Strong Inter-Stream Correlation

To achieve an effective global partitioning solution the *joint distribution across multiple streams* has to be considered. This joint distribution essentially determines the performance gain achievable by the multi-route solution. For our example query Q_m of Sec. 1, assume the media exposure of a company were correlated to its location instead of its industry, i.e., *independent* of the distribution of the stock stream. The news stream would be partitioned driven by location rather than by industry. In this case each partition of the news stream would have to be joined with every single partition of the stock stream (a full cartesian partition product), since the range of each news stream partition may overlap with any of the stock stream partitions. Therefore the multi-route solution would not be able to reap any benefit from the join pair pruning opportunity introduced in Sec. 1.

Worst yet, this would lead to the *combinatorial explosion* in the number of combinations formed by the partitions from each input stream. Each combination potentially leads to a partition query with a distinct query plan. This will cause prohibitively high optimization costs due to having to generate a plan for each partition query. Worst yet simultaneously having to maintain such a huge number of execution plans at run-time will introduce prohibitive execution overhead.

Next let us characterize the joint distribution property for the case when a large number of partition queries could be pruned.

First we formally define the joint distribution across input streams. Consider a query *Q* specified on a set of streams $\{S_1, S_2, ..., S_m\}$. S_i is a data stream whose values come from the universe $\mathcal{U}_i$ ($1 \leq$ i $\leq$ m). Without loss of generality, we assume each universe $\mathcal{U}_i$ is indexed by the set of integers $\{1, 2, ..., |\mathcal{U}_i|\}$.

Definition 5. *Given a combination of values* $(v_1, v_2, ..., v_m)$ $(1 \leq v_i \leq |\mathcal{U}_i|)$*, the joint frequency* $f(v_1, v_2, ..., v_m)$ *is the number of join results produced by* $v_1 \bowtie v_2 \bowtie, ..., \bowtie v_m$*. Then the joint distribution of input streams* $\{S_1, S_2, ..., S_m\}$ *is captured as a m-dimensional array (tensor), whose* i^{th} *dimension is indexed by the values of stream* S_i *(1≤i≤m) and whose cells contain the joint frequency of the corresponding combination of values.*

To avoid the generation of too many partition queries, the joint distribution must demonstrate strong correlation. The intuition comes from the characteristics of a static database table dt with strongly correlated attributes. Given a dt with two strongly correlated attributes X and Y with num_X and num_Y denoting the number of distinct attribute values of X and Y respectively, then the number of its distinct tuple values is typically far fewer than $num_X * num_Y$. In the extreme case when the values of X exactly determine one value of Y, the number of distinct tuple values of dt is equal to min(num_X,num_Y).

Therefore intuitively if the input streams show strong inter-stream correlation, the number of the non zero cells of the joint distribution tensor would not be large. Since only a non zero cell would potentially form a partition query, the number of partition query would thus not be large under this condition.

Furthermore this strong inter-stream correlation leads to the following important observation.

Observation 2. *The joint distribution of the tuples in each partition produced by CORBA is statistical uniform if the strong inter-stream correlation exists in the joint distribution.*

This observation can be justified as follows. First, the tuples in each partition share similar join selectivity when joining with the same partition, because the tuples within each partition formed by CORBA have similar statistics.

Second, the tuples in each partition have a high chance of having to join with the same partition. Given any two streams S and R, if S and R are highly correlated, a partition S_i of stream S only needs to join with one or at least very few partitions of R. This leads to the high possibility that the tuples in a given partition will all join with the same partitions.

Since the joint distribution across input streams represents the join relationships of the tuples in each stream, the above leads us to derive that the tuples in each partition formed by CORBA tend to share the similar joint distribution.

This observation enables us to search for the effective GP_j solution on the abstracted partition level joint distribution model rather than at the original tuple value level model per Def. 5. Therefore the search space of the GP_j is significantly reduced.

5.2 Inter-Stream Correlation Evaluation

In this section we propose a lightweight method to evaluate if the input streams of a given query show strong inter-stream correlation property. We first introduce the *Partition-Aware Join Graph* (PAJ) to model the joint distribution across multiple streams. The key idea is that the PAJ model is built on the uniform intervals produced by the *CORBA* algorithm.

Definition 6. *Let* $\mathbb{S}$ *represent a set of streams participating in query Q. A* ***PAJ*** *model of query Q and* $\mathbb{S}$ *is a graph* $G_{PAJ}(\mathcal{V},E)$ *such that (1) each node* R_i *in* $\mathcal{V}$ *represents a partition* $R_i \in R$ *with* $R \in \mathbb{S}$*, (2) an edge* $e_{ij} = (R_i, T_j)$ *exists in E if partitions* $R_i \in R$ *and* $T_j \in T$ *where* $R, T \in \mathbb{S}$ *form a join pair in Q potentially producing output, (3) each node* $R_i \in \mathcal{V}$ *is annotated with a label tc(*R_i*) =* $|R_i|$ *denoting the cardinality of* R_i*, (4) each edge* (R_i, T_j) $\in E$ *records the cardinality of its estimated join output denoted by jc(*R_i*,*T_j*) =* $|R_i \bowtie T_j|$.

Similar to [15] a *PAJ* model encodes a partitioning of the stream data along with aggregate information about its statistics and the join relationships among the partitions. Consider a specific *PAJ* model, say the graph G_{PAJ}, and a partition $R_i \in R$. Let $T_j \in T$ be a partition connected to R_i with $(R_i, T_j) \in$ E. Then we define *jr(*R_i*,*T_j*) = jc(*R_i*,*T_j*)/(tc(*R_i*)* as the join ratio between R_i and T_j. In this case, each tuple in R_i is expected to join with each tuple in T_j with the probability of *jr(*R_i, T_j*)*.

Example 1. *Consider an example query Q1 over the partitioned streams R, S, T, and U after join pair pruning shown in Fig.6.*

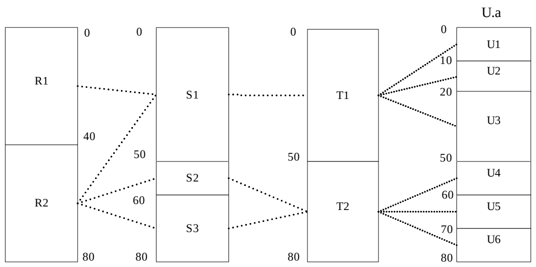


Figure 6: Horizontal Partitioning

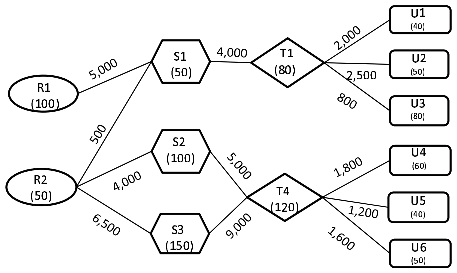


Figure 7: Original Partition-Aware Join Graph

```
Q1: SELECT * FROM R, S, T, U
    WHERE R.a = S.a and S.a = T.a and T.a = U.a
    WINDOW 60 seconds
```

Fig. 7 shows the PAJ model of Q1. In this graph, node R_1 represents one partition of R with $tc(R_1)$ = 100 samples falling in this partition. Accordingly node S_1 represents a partition of S with cardinality 50. Their join operation denoted by the edge (R_1, S_1) is estimated to produce $jc(R_1, S_1)$ = 5000 results.

Based on the PAJ model we now propose a lightweight method to measure the inter-stream correlations with the notion of *mutual information* [3]. Given two random variables X and Y, *mutual information* is a common technique used to measure their dependency. This metric calculated with Eq. 4 is also widely adopted to measure the correlation of attributes pairs of a static database table.

$$I(X,Y) = \sum_x \sum_y P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{4}$$

In Eq. 4 $P(x, y)$ is the *joint probability distribution function* of X and Y, and $P(x)$ and $P(y)$ are the *marginal probability distribution functions* of X and Y respectively. The mapping between the streams and the random variables is established as follows using query Q_1 as example.

Let $\mathbb{T}$ and $\mathbb{U}$ denote a set of nodes in G_{PAJ} corresponding to streams T and U respectively. First we map the stream T to a random variable X and accordingly each node T_i in $\mathbb{T}$ to one value of *X* denoted as x_{T_i}. Similarly let *Y* be the random variable modeling stream U and node U_j in $\mathbb{U}$ be mapped to y_{U_j} of variable *Y*. $P(x_{T_i})$ is defined as the percentage of the number of tuples in T_i over the total sampling cardinality of stream *T*. It is calculated as:

$$P(x_{T_i}) = \frac{tc(T_i)}{\sum\limits_{T_j \in \mathbb{T}} tc(T_j)} \tag{5}$$

Then the joint probability $P(x_{T_i}, y_{U_j})$ of x_{T_i} and y_{U_j} is defined as the percentage of the number of results produced by joining T_i with U_j over the expected total output produced when joining streams T and U, namely:

$$P(x_{T_i}, y_{U_j}) = \frac{jc(T_i, U_j)}{\sum\limits_{T_j \in \mathbb{T}, U_k \in \mathbb{U}} jc(T_j, U_k)} \tag{6}$$

Given this *mutual information* metric for pair streams, we use the *average mutual information* (*MI*) to assess the overall inter-stream correlation across all streams involved in a query *Q*. Given a set $\mathbb{S}_p$ of all stream pairs (S_i, S_j) with edges connected in G_{PJ}, $\mathbb{V}_p$ is a set containing the random variable pairs (X, Y) mapped from the stream pairs in $\mathbb{S}_p$. *MI* is calculated as:

$$MI = \frac{\sum\limits_{(X,Y) \in \mathbb{V}_p} I(X,Y)}{|\mathbb{V}_p|} \tag{7}$$

As confirmed by our experiments in Sec. 6.3 *MI* in Eq. 7 effectively indicates the potential gain of the multi-route solution. The larger *MI* is, the better the performance of the multi-route solution will be. When *MI* is greater than some constant value, the multi-route strategy is guaranteed to win against the single plan strategy.
Early Termination. With the strong *inter-stream correlation* and *skewed uniformity* properties we now are ready to design an *early termination* mechanism for short-cutting the multi-route optimization. Namely if the input streams are found to not exhibit either skewed uniformity or strong inter-stream correlation, then we propose to terminate the expensive multi-route optimization procedure. This decision criteria, albeit simple, enables *CMR* to quickly converge to the traditional "single plan" approach in any multi-route unfriendly environment.

5.3 CONCUR Algorithm

Guided by the *MI* metric, we now demonstrate the CONCUR algorithm at improving the partitioning by clustering and merging nodes with similar joint distributions. This algorithm leads to a reduced number of the final partition queries and in turn an increased *MI*, indicating the run-time execution performance has improved.

Algorithm 3 CONCUR(P[n][ ], ct)

```
Input: partitioning of stream $S_1, S_2, ...S_n$ P[n][ ], clustering threshold $ct$
Output: PAJ $G$
1: Initialize PAJ with one node per partition in P[n][ ];
2: for stream $S_i$ from $S_1$ to $S_n$ do
3:     while change do
4:         $\mathbb{P} = map(S_i)$;
5:         Merge(cluster($\mathbb{P}$,$ct$));
6:     end while
7: end for
```

We now describe the overall process flow of the *CONCUR* algorithm (Alg. 3). *CONCUR* receives as input the partitions of the n input streams of query *Q* denoted by P[n][ ], and a clustering error threshold ct. It returns a compressed *PAJ* graph *G*. *CONCUR* first initializes the *PAJ* graph with the given partitions of each stream as nodes and the join pairs as edges (Line 1).

Then *CONCUR* utilizes a clustering algorithm to identify the nodes with similar statistics. Given a set $\mathbb{R}$ containing the nodes R_i in stream R. Inspired by [15] we measure the similarity of nodes by mapping every node R_i of $\mathbb{R}$ to a multi-dimensional point (Line 4). More specifically we use set *links*($\mathbb{R}$) to represent the nodes with a direct edge linked to any node in $\mathbb{R}$ in the *PAJ* model. $|Links_{\mathbb{R}}|$ denotes the cardinality of set *links*($\mathbb{R}$). Then every node R_i of $\mathbb{R}$ is mapped to an $|Links_{\mathbb{R}}|$-dimensional point, where each dimension corresponds to a node R_{link} in *links*($\mathbb{R}$). This coordinate value

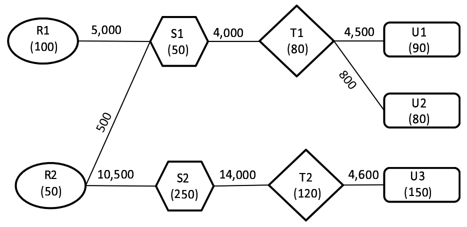


Figure 8: Compressed Partition-Aware Join Graph of Q1

of R_i for node R_{neigh} is set to *jr(R_i,R_{link})*. In other words, the multi-dimensional point of R_i indicates the other nodes to which R_i is linked and their corresponding join ratios.

We thus can measure the similarity of two nodes in terms of the distance in this multi-dimensional space. The nodes with similar characteristics can be identified by applying any clustering algorithm (Line 5). In this work we adopt BIRCH [21] with the *diameter metric* (Eq. 8) as the tightness measure of a cluster.

$$diameter(\mathbb{R}^c) = \sqrt{\frac{\sum_{i=1}^{|\mathbb{R}^c|}\sum_{j=1}^{|\mathbb{R}^c|}(R_i - R_j)^2}{\mid \mathbb{R}^c \mid (\mid \mathbb{R}^c \mid -1)}} \qquad (8)$$

After a cluster $\mathbb{R}^c$ is identified, *CONCUR* uses the merge function to substitute all nodes in $\mathbb{R}^c$ with a single new node R_{new} that represents the union of all partitions in $\mathbb{R}^c$ (Line 4). The statistics of the new node R_{new} are naturally defined as: *tc(R_{new}) = $\sum_{R_i \in \mathbb{R}^c}$ tc(R_i)*, and *jc(R_{new},t) = $\sum_{R_i \in \mathbb{R}^c}$jc($R_i$,t)*.

Example 2. *In Fig. 7, node S_2 is abstracted as a point $p_2(40, 50)$ with 40 and 50 being $jr(R_2, S_2)$ and $jr(S_2, T4)$. Similarly node S_3 is represented by the point $p_3(43, 60)$. According to Eq. 8, the diameter($\{p_2, p_3\}$) is measured to be 10. Given an error threshold ct = 20 > 10, the nodes S_2 and S_3 are declared to be similar enough to be substituted by a new node S_{new} with $tc(S_{new}) = 250$, $jc(S_{new}, R_2) = 10500$, and $jc(S_{new}, T_2) = 14000$.*

This process (Lines 4, 5) continues until every stream has been examined at least once and no nodes remain that form a valid cluster under the restriction of the error threshold *ct* (Lines 2, 3). After sequentially merging $\{U_1, U_2\}, \{U_4, U_5, U_6\}$, and $\{S_2, S_3\}$, the final partitioning is derived as shown in Fig. 8.

6. EXPERIMENTAL EVALUATION

6.1 Experimental Design

Experimental Setup: All our experiments are run on a machine with Java 1.6.0.22, Windows 7 with Intel(R) Core(TM) i7 CPU @2.67GHz processor and 4GB of RAM.

We compare the performance of our *CMR* against the three key alternative approaches in the literature, namely (1) the traditional "single plan for all data" system [18] (SP), (2) the "query mesh" system [13] (QM), and (3) the "Sharing-Aware Horizontal Partitioning" [20] (HP). For SP, we implement a multi-way join (MJoin) [18] operator. MJoin is a generalization of the symmetric binary join algorithm shown to provide the best plan for each stream. HP [20], although focused on static databases, is the state-of-the-art work in horizontal partitioning. We apply this to our stream context by adding sampling for collecting the data statistics. We instantiate HP's query plans in our execution infrastructure. The *QM* strategy [13] with a content-driven start solution is the approach in the literature closest to our *CMR*, since both conduct content-based partitioning. However we replace their simplistic partitioner heuristic by our more sophisticated CMR optimizer.

To ensure a fair comparison, all systems are implemented in the CAPE platform [5]. Each implementation uses as much of the same code base as possible. Since QM, HP, and CMR are implemented within the same software framework, we can employ the same system parameters (both for optimization and execution parameters) as shown in Table 1.

Parameter	Value	Description
Data Arrival	*Poisson*	Data arrival distribution
μ	500 msec	Mean inter-arrival rate
$\|A\|$	6	# of attributes in tuple schema
$\|T_{dq}\|$	1,000	Maximum # of tuples dequeued by an operator at a time
W^{TC}	1,000 tuples	Classification window size
Ruster size	100 tuples	Minimum *ruster* size

Table 1: Defaults used in the experiments.

Metrics. We compare *CMR* against its competitors by measuring: a) the average output rate at run-time, b) the execution time with varying correlations at run-time, c) the cumulative number of tuples produced over time, and d) the optimization time with varying correlations at compile time.

Key Features. We study the effectiveness of the key features of CMR : 1) the partitioning number threshold in the *CORBA* algorithm, and 2) the CORBA and CONCUR algorithms.

6.2 Data Sets and Queries

Weather Dataset: This real world dataset made available by *CDIAC* [4] consists of weather measurements organized by month and collected over several years by thousands of weather stations. For our experiments, we chose the readings in the month of November over ten consecutive years (2000, 2009). The attributes of interest were the latitude, longitude, and brightness information. The latitude and longitude pinpoint the location of the weather station taking the readings. To explicitly represent the physical proximity of weather stations we divided locations on the earth into square grids consisting of 10 degrees of latitude and longitude each. We then replaced the location attributes of each event with the ID of the grid cell that it falls in. The brightness information is a (0,1) score indicating whether the illuminance criterion was satisfied.
Soccer Dataset: This real world sport stream [12] is used as challenge benchmark at DEBS 2013. It is produced by the Real-Time Locating System deployed on a soccer field of Nuremberg Stadium in Germany. Data originates from sensors located near the players' shoes and in the ball. Every event describes the position of a given sensor in a three-dimensional coordinate system. We have divided the sensor readings into four streams (Team A, Team B, the ball, and the referee) based on the identity of the people that the sensor is located on. We have equally divided the soccer field into 4 areas and replaced the coordinate attributes of each event with its corresponding area. This facilitates the analysis of the defense and offense relationship among the players.
Berkeley Dataset: The third real dataset is composed of readings from sensors in the Intel Research, Berkeley Lab [14] between Feb. 28th and Apr. 5th, 2004. We have partitioned these sensor readings into five data streams based on sensor locations. Each stream corresponds to a group of sensors in close proximity to each other.

Synthetic Datasets. Beyond the rich sets of real data we also work with synthetic datasets. By manipulating the parameters of synthetic data we evaluate the performance patterns of CMR under varying data skewness and distribution. To establish data skew, we employ the *Uniform* and *Poisson* distributions with parameter variations as per Table 2.

Queries: We deploy N-way join queries, as N-way join queries are among the core and most expensive queries in database systems. Such queries are commonly used to discover correlations across data from different sources and to compose complex results. In the context of the *weather data* the join query is performed on the cell and brightness attributes of streams to identify weather stations that are located physically near each other yet are observing different brightness measures. Such information might be valuable for the analysis of climate change. For the *soccer data*, we performed a join on the four streams using the area attribute to identify sensors (players) located physically near each other. Such a join query could potentially be used to analyze the defense and offense relationships among the players. For the *Berkeley data*, the join query performed on the temperature attribute may help to detect fire "hotspots" or support automatic temperature control inside buildings. The sliding window on the join operator enforces that the matched readings are taken at nearby points in time. For synthetic data experiments unless otherwise stated we use an equi-join of 5 streams, i.e., $S_0 \bowtie S_1 \ldots \bowtie S_4$.

Data Distributions		
Name	**Parameters**	**Application Examples**
Uniform	$\alpha \in \{\ldots,\beta\text{-}1,\beta\}$ $\beta \in \{\alpha,\alpha+1,\ldots\}$ X $\in \{\alpha,\ldots,\beta\text{-}1,\beta\}$	● Long-term patterns of data ● Distribution of moving objects in some geographic areas
Poisson	$0 < \lambda < \infty$ X $\in \{0,1,\ldots\}$	● # people at a counter ● # of times web server accessed per minute
Uniform (α = 0, β = 100): *min*: 0.0, *max*: 100.0, *med*: 49.0, *mean*: 49.7, *ave.dev*: 25.2, *st.dev*: 29.14, *var*: 849.18, *skew*: 0.05,		
Poisson (λ = 1): *min*: 0.0, *max*: 7.0, *med*: 1.0,*mean*: 0.97, *ave.dev*: 0.74, *st.dev*: 1.01, *var*: 1.02,*skew*: 1.17, *kurt*: 1.89 *Distribution transitions*: (λ = 1)$\rightarrow$(λ = 3)$\rightarrow$(λ = 5)...		

Table 2: Distribution statistics for synthetic Data.

6.3 Comparing Alternative Solutions

Average Output Rates. First we compare *CMR* to SP, QM, and HP with both uniform and poisson distributed synthetic data (settings in Table 1) to verify the scope of the applicable scenarios for multi-route solutions. The results are averaged over 10 runs of 10 minutes each to measure the average output rate in Fig. 9.

First let us discuss the worst case scenario for the multi-route solution, namely, when the data is uniform. We observe that in this case, *CMR* tends to default to a solution with a single route per stream or occasionally at most two routes. This is expected as with a none-skewed dataset, no benefit can be gained from distinct routes. It is in fact desirable as we would not enforce any additional routes. In this case, we observe on average *CMR, QM, and HP* are 2.2% worse than SP in output rate due to the extra overhead of the multi-route execution infrastructure (left in Fig. 9). CMR, QM, and HP exhibit similar performance due to the same execution infrastructure they adopt.

In the second experiment we instead utilize poisson distributed streams. In general the poisson distributed data is skewed. Yet the adjacent values of the poisson distribution tend to show similar statistics. This benefits the multi-route solution. As shown in

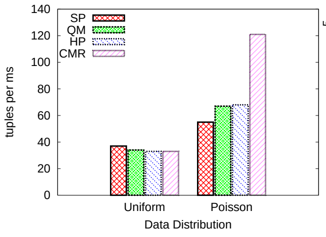


Figure 9: Output rates

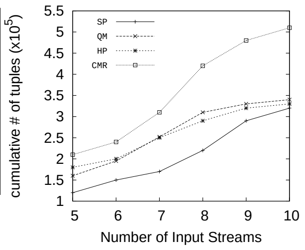


Figure 10: Stream number

the right side of Fig. 9, *CMR* on average has an 87% higher output rate than HP, 90% higher than QM, and 121% higher than SP. This confirms the effectiveness of the correlation-aware partitioning strategy of CMR when handling skewed data.

In summary our results show that *CMR* has a low overhead as there is no significant degradation in performance if datasets are not skewed, yet exhibits significant improvements where skew arises.

The Effect of Varying Query Complexity On Execution. Next we evaluate the performance of CMR by varying the number of streams involved in the join queries. In this experiment we use the real life **weather data** (Sec. 6.2). The number of streams varies from from 5 (2000 to 2004 weather data) to 10 (2000 to 2009 weather data). Here we analyze the total number of tuples created after 5 minutes (averaged over 10 runs).

As shown in Fig. 10 CMR outperforms SP at least by 60 percent in all cases. This is due to the consistent existence of strong correlations as shown below. The readings produced by the sensors located in the same cell at nearby points in time tend to have similar brightness measures. This leads to the *skewed uniformity* shown in each weather stream. Preserving this correlation is straightforward when mapping the categorical weather station identity to numerical domain, because the weather stations are already ordered by their locations in the original data files. Furthermore the weather streams share the similar statistical patterns each year. Therefore the weather streams of different years are strongly correlated no matter how many weather streams are involved in the query.

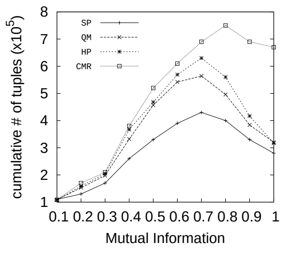
We also measure the MI value in this experiment. It varies in a small range from 0.55 to 0.61. Therefore this experiment also confirms that the performance of CMR strongly relies on the intensity of the correlations instead of on the complexity of the queries.

Furthermore we observe that HP and QM *no longer* show much advantage over SP as the query length increases to 9 or beyond. This might be caused by their ignorance of the optimization opportunities hidden behind the strong inter-stream correlations. These opportunities increase as the number of input streams increases.

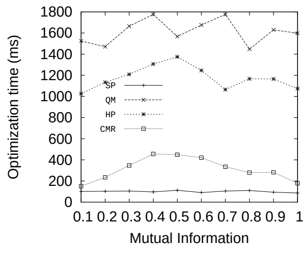
In this experiment we also evaluate our Obs. 2 proposed in Sec. 5.1. The weather stream of each year is divided into 11 to 13 uniform intervals by CORBA. We then produce the optimal query plan for each single tuple based on the statistics collected by CORBA. Most of the tuples (86% in average) within each uniform interval are observed to share same optimal plan and hence show similar joint distribution statistics. This confirms the validity of Obs. 2.

The Effect of Varying Data Correlations on Execution. Next we evaluate the effectiveness of CMR for data streams with increasing data correlations. We control the data correlations by varying *MI* from 0.1 to 1. The stream generator achieves such MI variations by adjusting the percentage of each tuple value over the data stream.

Fig. 11 (a) shows the total number of tuples created after 5 minutes (averaged over 10 runs). For weakly correlated data (*MI* $<$

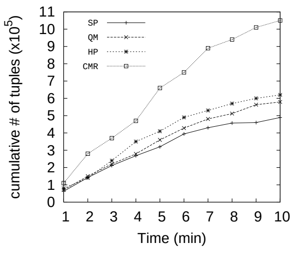


(a) Number of tuples produced

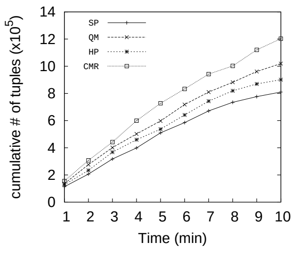


(b) Optimization time

Figure 11: Data correlations



(a) Soccer data



(b) Real sensor data

Figure 12: Total number of tuples produced over time

0.2), all four systems yield almost the same number of output tuples. As the data becomes more correlated (*MI* $\geq$ 0.2) CMR, QM, and HP become faster than SP. At *MI* = 1, we observe that CMR is 3-fold faster than SP. As the *MI* rises to 5 (not shown in this chart due to the scope restriction of the x axis) CMR is 10-fold faster than SP. This experiment confirms that *MI* effectively indicates the potential gain achievable by applying CMR instead of traditional single plan technology. However when the data is strongly correlated (MI $\geq$ 0.9), HP and QM no longer clearly outperform SP. In this case both HP and QM do not utilize the join pair pruning opportunities nor succeed to merge the partition combinations to a small number of partition queries (see Sec. 5), while stronger correlations indicate that more such opportunities should exist for optimization. Such opportunities are indeed leveraged by CMR.

The Effect of Varying Data Correlations on Optimization. Next we evaluate the optimization time using the above setting. As shown in Fig.11 (b) although all three multi-route algorithms are more expensive than SP as expected, CMR is consistently superior to QM and HP. When data is weakly correlated, CMR exhibits similar optimization time with SP. This can be explained by our early termination mechanism that quickly determines whether a multi-route solution should be even explored. For highly correlated data, CMR benefiting from the small number of partition queries, greatly outperforms QM and HP. In all cases CMR significantly outperforms QM and HP, because CMR successfully decomposes the stream partitioning and query planning. HP is about 30 percent faster than QM as the latter applies the traditional algorithm [18] to compute the optimal join plans separately for each partition query.

Total Number of Tuples Produced Over Time. We use two real datasets (*soccer data* and *Berkeley data* introduced in Sec. 6.2) to evaluate the total number of tuples produced by the four strategies over time. Here we display the average output for the first 10 minutes of longer execution runs.

The **soccer data** shows strong correlations in the sense that our MI metric is measured as 0.76. The reason is that the positions of the players are highly correlated with the specific areas on the field which they cover. For instance the strikers usually cover the offensive half of the field, while the goalkeepers generally do not leave the penalty box. These strong correlations between the sensor Id (players) and the area attributes indicates that the players in the same position tend to cover similar areas. This leads to the *skewed uniformity* shown in the player streams. When performing the intra-stream partitioning (CORBA) the identities of the players are mapped to consecutive numerical values along their original order in the metadata file of soccer data [12]. Since in the metadata file the names of players are naturally ordered by their positions, this property is preserved after the mapping. Furthermore the referee stream and the ball stream are also highly correlated, since in general the referee is physically close to the location of the ball.

As shown in Fig. 12 (a), *CMR* significantly outperforms other alternatives. This is because that CMR successfully discovers the uniform intervals of the player streams and fully utilizes the strong inter-stream correlations between referee stream and ball stream. In the SP approach, single plan optimization coarseness leads to producing a lot more intermediate results. These results gradually fill up the queues and hinder the performance of the system. Although compared to SP, QM and HP solutions show better performance, both of them suffer from lack of an effective partitioning strategy. They either produce too many partitions, or incorrectly group the tuples without sharing similar optimal plans into the same partition. This makes them worse than CMR by more than 50 percent.

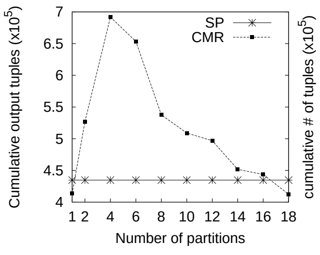
Similar to the soccer data experiment, for the **Berkeley data** CMR significantly outperforms the other three approaches as shown in Fig. 12 (b). This performance gain originates from the strong correlations between the location of the sensors and their temperature readings. This leads to a relatively high MI value as 0.42 – high enough to provide CMR with opportunities to discover good partitions. When conducting the intra-stream partitioning (CORBA), the sensor tags are mapped to consecutive positive integers by following their original order in the metadata file. This mapping successfully keeps this correlation, since the sensors in physical proximity are indeed adjacent to each other in the metadata file.

Similar to the *weather stream* experiment, Obs. 2 of Sec. 5.1 is also evaluated in this set of experiments. CORBA divides each soccer stream into 3 or 4 partitions and each Berkeley stream into 5 to 7 partitions. More than 80% percent of tuples in each partition are observed to share same optimal plan. This again confirms the validity of Obs. 2.

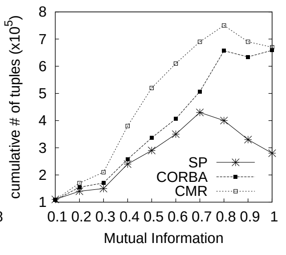
6.4 Key Features of CMR

The Effect of the Partition Number Threshold. The threshold τ which restricts the maximum number of partitions made on each stream is used to identify if the distribution exhibits the *skewed uniformity* property (Sec. 4.2).

Fig. 13 (a) shows the number of tuples produced in 5 minutes by CMR and SP when varying the number of partitions per input-stream. When the number of partitions is smaller than 16, CMR processes more tuples than SP. As each individual stream is divided into more than 16 partitions, CMR becomes worse than SP due to having to maintain a larger and larger number of partition queries and their routes. Clearly the benefit of supporting multiple plans is outweighed by the overhead introduced by route maintenance and tuple classification. Given this empirical evidence we use this calibrated value of 16 as the partition number threshold τ for our experiments. This threshold is used by CORBA to evaluate whether an input stream shows skewed uniformity.



(a) Partition threshold



(b) Partition Algorithms

Figure 13: Impact of different features.

The Effect of CORBA and CONCUR. Next we evaluate the effectiveness of our *CORBA* and *CONCUR* algorithms independently. The effectiveness of *CORBA* is evaluated by excluding *CONCUR* from CMR. Then the performance of *CONCUR* can be naturally demonstrated by comparing the *CORBA* only optimizer against the full CMR optimizer.

Fig. 13 (b) depicts the number of output tuples produced after 5 minutes by the *full* CMR, the *CORBA* only CMR, and SP. The difference is shown over increasing data correlations. We vary the data correlations by the same approach applied in the correlation experiments of Sec. 6.3. As Fig. 13 (b) shows, *CMR* processes tuples up to 50 percent faster than *CORBA*, because *CONCUR* eliminates unnecessary partitions who have similar joint distributions with others. However for weakly correlated (MI < 0.2) and highly correlated (MI ≥ 0.9) data, *CMR* does not show much advantage over *CORBA*. Only in these settings, each stream does not produce many local partitions. Hence less opportunities for partition merging are offered to *CONCUR*. *CORBA* outperforms *SP* in all cases with MI ≥ 0.2. This is because that CORBA successfully groups tuples sharing similar query plans into the same partition. In summary this experiment confirms that both *CORBA* and *CONCUR* algorithms contribute to produce effective partitioning.

6.5 Summary of Experimental Results

Our main findings can be summarized as: 1) *CMR* improves execution time and output rate metrics by up to 10-fold compared to the single plan solution. 2) The correlation-aware partitioning strategy enables *CMR* to achieve effective partitioning and dramatically outperforms the existing partitioning strategies, both QM [13] and HP [20]. 3) In the worst case when the data is uniform and thus a traditional single route solution would have been ideal, *CMR*'s performance is only 2% slower than that of SP. All in all these experiments demonstrate that *CMR* achieves significant performance improvements over alternative solutions. Thus indeed it is a promising optimizer for multi-route stream query processing.

7. CONCLUSION

In this paper, we have proposed a practical multi-route optimizer called *CMR*. By carefully analyzing the stream data properties revealed in the multi-route strategy winning scenario, *CMR* successfully decomposes the stream partitioning and query planning with an effective correlation-aware partitioning strategy. The layered structure of CMR makes it amendable to handling characteristics drift which frequently arises in dynamic streaming context. All the above features make *CMR* a practical and effective optimizer for time-critical applications.

8. REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik. The design of the borealis stream processing engine. In *CIDR*, pages 277–289, 2005.

[2] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *SIGMOD*, pages 261–272, 2000.

[3] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. on Information Theory*, 14:462–467, 1968.

[4] R. Eastman and S. Warren. Extended edited synoptic cloud reports from ships and land stations over the globe, 1952-2009 (ndp-026c).

[5] E.Rundensteiner and L.Ding and T.Sutherland and Y.Zhu and B.Pielech and N.Mehta. Cape: Continuous query engine with heterogeneous adaptivity. In *VLDB*, pages 1353–1356, 2004.

[6] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. Spade: the system s declarative stream processing engine. In *SIGMOD Conference*, pages 1123–1134, 2008.

[7] O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. In *Studies in Complexity and Cryptography*, pages 68–75. 2011.

[8] N. B. Herodotos Herodotou and S. Babu. Query optimization techniques for partitioned tables. In *SIGMOD Conference*, pages 49–60, 2011.

[9] P. Indyk, R. Levi, and R. Rubinfeld. Approximating and testing k-histogram distributions in sub-linear time. In *PODS*, pages 15–22, 2012.

[10] M. Klazar. Bell numbers, their relatives, and algebraic diff. equations. *J. Comb. Theory Ser.*, pages 63–87, 2003.

[11] S. C. Mehul A. Shah, Joseph M. Hellerstein and M. J. Franklin. Flux: An adaptive partitioning operator for continuous query systems. In *ICDE*, pages 25–36, 2003.

[12] C. Mutschler and F. IIS. The dataset of debs 2013 grand challenge. *http://www.orgs.ttu.edu/debs2013*.

[13] R. V. Nehme, K. Works, C. Lei, E. A. Rundensteiner, and E. Bertino. Multi-route query processing and optimization. *J. Comput. Syst. Sci.*, 79(3):312–329, 2013.

[14] P. Bizarro and S.Babu and D.DeWitt and J.Widom. Content-based routing: Different plans for different data. In *VLDB*, pages 757–768, 2005.

[15] N. Polyzotis. Selectivity-based partitioning: a divide-and-union paradigm for effective query optimiation. In *CIKM*, pages 720–727, 2005.

[16] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, pages 486–495, 1997.

[17] S.Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *SIGMOD*, pages 407–418, 2004.

[18] S.Viglas, J.Naughton, and J.Burger. Maximizing the output rate of multi-way join queries over streaming inf. sources. In *VLDB*, pages 285–296, 2003.

[19] TradingMarkets. http://www.tradingmarkets.com/.

[20] K. Tzoumas and et.al. Sharing-aware horizontal partitioning for exploiting correlations during query processing. *PVLDB*, 3(1):542–553, 2010.

[21] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD*, pages 103–114, 1996.