2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

GPS: A General Framework for Parallel Queries over Data Streams in Cloud

Xiaoyong Li, Yijie Wang, Yue Zhao, Yuan Wang, Xiaoling Li
Science and Technology on Parallel and Distributed Processing Laboratory, College of Computer
National University of Defense Technology
Changsha, Hunan, P. R. China, 410073
E-mail: {sayingxmu, wangyijie, zhaoyue, purplego, lixiaoling}@nudt.edu.cn

***Abstract*—Parallel query processing over data streams in cloud computing environments has attracted considerable attention recently in various fields, due to the huge potential value of analyzing massive data or big data in a large number of streaming applications. Nevertheless, existing studies on queries primarily focus on the algorithms for the specific query types with the lack of the general framework for processing various queries. Moreover, existing parallel frameworks in cloud such as MapReduce and its variations are not suitable for many complex queries over complex data streams. In this paper, we extensively discuss the problem of designing the general framework for parallel queries over data streams in cloud. Particularly, we propose and implement a framework called GPS, which can be well adapted to various queries over complex data streams like the uncertain data streams. Furthermore, we further propose a hierarchical and general parallel model for queries over data streams based on the proposed framework, which is more flexible than the MapReduce model. The skyline queries over uncertain data streams based on our proposed framework with real deployment are conducted as an example to verify the performances of our proposals.**

***Keywords*—parallel query; parallel framework; data streams; cloud computing; skyline queries**

I. Introduction

Many complex queries over massive data or big data have received considerable attention recently with the emergence of a large number of practical applications in domains like sensor network, RFID network, data cleaning, location-based service, network traffic analysis, market surveillance and traffic monitoring with radar. Moreover, the data in these scenarios are often generated dynamically and continuously, even with uncertainty [1], [2]. For example, in the application of hazardous weather monitoring with radar networks [3], a large amount of meteorological data are collected continuously and fed to the real-time processing system to predict natural disasters like tornados and severe storms [4]. In all these scenarios, it is highly desirable to conduct timely data analysis over the complex data streams.

In the complex streaming environments, the challenge of the query mainly lies in various aspects, such as the computational complexity, the streaming complexity, and the query requirements. First, the computation of the query, such as probabilistic skyline query [5] and nearest-neighbor query [6], is CPU-intensive, which requires powerful processing capability. Second, the fast arriving rate (e.g., 200Mb/sec from a single radar node [4]), as well as the limited storage space forces us to seek efficient algorithms over sliding windows. Third, the strict limit of the response time and the large sliding window concerned by the users make the query more challenging. Nevertheless, the existing centralized query techniques [7], [8] are hard to meet the query requirements, which spurs us to explore the parallel processing techniques for the complex queries.

Cloud computing as a new emerging distributed computing paradigm driven by economies of scale, provides abundant computing resources to users with minimal management effort in the data centers. Cloud is a platform or an infrastructure that enables execution of codes (services, applications), in a managed and elastic way. In computational view, cloud computing is actually a network of data centers and is described as a powerful, low-cost, and energy-efficient approach to future computing [9], which has the potential to yield fast response times and permit processing of high-throughput input streams.

There has been an increasing interest in parallelizing some specific queries (e.g., skyline queries) in various distributed and parallel environments recently [10], [11]. All the existing approaches share the idea of partitioning the whole dataset to subsets, processing locally the subsets in parallel, and finally merging the results. The correctness of the approaches stems from the additivity of the query operator, which means that $Q(S_1 \cup \cdots \cup S_n) = Q(Q(S_1) \cup \cdots \cup Q(S_n))$, where $Q(S_i)$ denotes the queries over S_i.

However, all the existing studies on queries over data streams are all devoted to the research on a specific query type with the lack of a general framework for various queries. Although many parallel frameworks based on the cloud computing environment are proposed, such as MapReduce [12], HOP [13], Twister [14], HaLoop [15] and Hadoop++ [16]. These frameworks are all not well adapted to the parallel queries over data streams, which mainly stems from the following facts: 1) The frameworks are mainly designed for the parallel processing for various tasks over static data based on the distributed file systems, but not for the queries over data streams without any file management systems; 2) The tasks in many complex queries cannot be partitioned into many individual and independent computational tasks, especially for the queries that do not satisfy the additivity [2], such as the uncertain skyline top-k queries; 3) Many complex queries need interactions of the parallel nodes or even need optimizations with the assistant nodes, but the MapReduce based frameworks can not well support the communication of the parallel nodes.

Motivated by these, we study the problem of parallel query

978-0-7695-5088-6/13 $31.00 © 2013 IEEE
DOI 10.1109/HPCC.and.EUC.2013.161

processing over data streams in cloud computing environments. Specifically, we propose a **G**eneral framework for **P**arallelizing **S**treaming queries called *GPS*. To summarize, we provide the following contributions in this paper:

- For the first time, we propose a general framework GPS for parallel queries over data streams with the window-based data stream model;
- We further propose a hierarchical and general parallel model for queries over data streams based on the GPS;
- We conduct the experiments of the skyline queries over uncertain data streams based on our proposed parallel framework with real deployment as an example to verify the effectiveness and efficiency of our proposals.

The remainder of this paper is organized as follows: Section II presents the proposed framework GPS in detail, which includes the introduction of the GPS architecture, and discussion of the optimization from various aspects. Section III describes the hierarchical and general parallel model based on the GPS, which is exclusively designed for parallel queries over data streams. We extensively evaluate the performances of our proposals from many aspects in Section IV. Finally, we conclude the paper and outline our future work in Section V.

II. General Parallel Framework

A. Architecture of GPS

To realize the parallel processing for the queries over data streams, we first propose a general parallel framework for the queries, and the overall structure is summarized in Fig. 1.

From Fig. 1, we can see that there are several layers and modules in the framework. The bottom is the fundamental communication platform, which is mainly responsible for the communication of the nodes involved in the parallel processing. The platform can be implemented with various middlewares, and in our prototype we adopt Ice*, which is an object-oriented middleware. The upper layer is the module for the input and management of data streams, which mainly consists of three submodules, including the probing module, the preprocessing module, and the caching and storage module. The probing module is mainly responsible for probing the streaming rate and throughput to optimize the later query processing. Preprocessing mainly includes the work of data cleaning, transformation and loading to adapt to the queries. The cache and storage module is responsible for optimizing the access and storage of the streaming data.

Based on the underlying management of the streaming data, we can adopt various models for data stream analysis. The most popular are the window-based models, which mainly contains the landmark model, sliding window model and snapshot model, in which the sliding window model is the most frequently used. Moreover, according to criteria of the physical and logical model, as well as updating time interval, the sliding window model can be divided into count-based sliding window and time-based sliding window, as well as tumbling window, sliding window and jumping window [17], respectively.

*Zeroc. Available: (http://www.zeroc.com/)

After defining the models for data stream analysis, we can develop the module for the parallel processing for various queries, which is the most important part in the whole framework. To realize the parallel processing, we propose a hierarchical and general parallel model, which will be introduced in detail in the next section. Furthermore, to improve the parallel query processing, several optimization modules are developed in the framework, including the indices, nodes organization, load balancing, and fault tolerance. The indices mainly include three aspects: 1) the index for streaming items maintained in the windows, 2) the index for the nodes involved in parallel computing, and 3) the index for the data structures that defined by users to improve the queries. Generally, we can improve the queries over data streams by optimizing the organization of the parallel nodes in two ways, i.e., considering the network structure with various roles of the nodes and the data correlation for parallel nodes.

In GPS we try to develop the load-balance strategies in three aspects: 1) adjusting the input of streaming data, 2) controlling the distribution of streaming data, and 3) optimizing the window partition according to the capabilities of the nodes. The fault-tolerant module is exclusively designed to solve the problems resulted from the failures of the nodes or links. To explore the strategies we can mainly adopt the replication technology and coding techniques.

Finally, according to the specific definitions of various queries over data streams, we can develop different optimization strategies for the queries. Note that, the framework can support almost all kinds of queries over data streams for the users, including not only the query styles as ad hoc queries and the continuous queries, but also the specific queries like skyline queries, top-k queries, nearest neighbor queries, range queries, key-word queries, aggregate queries, join queries, and frequent items queries over data streams.

B. Considerations for Optimization

According to the introduction of the parallel framework addressed before, we can see that there are many problems needs to be further considered and solved, which are also the directions of our current and future work.

1) Size of Input Streaming Data Block: The characteristics of data streams such as real-time response and single-pass make a huge challenge to the queries over data streams. Besides, data streams exhibit unpredictability as they may be sometimes slow, but sometimes very fast with high throughput. Therefore, we need to analyze the blocks of the incoming items according to the real states of the streams, so as to process the streaming data timely and accurately. Thus, to avoid the overflow of memory or the incorrectness deriving from the overflow of the items, we need to take fully into account the size of input streaming data block and adaptively adjust the size according to the capability of the parallel system.

2) Load Balancing for All Nodes: Load balancing is one of the most important goals of the distributed systems, which aims to avoid the case that too heavy a load in some of the nodes while the other nodes are idle. Studying the load balancing strategies not only can avoid the performance degradation or a single point of failure, but also can improve the resource utilization. Thus, we need to consider the loads of the nodes

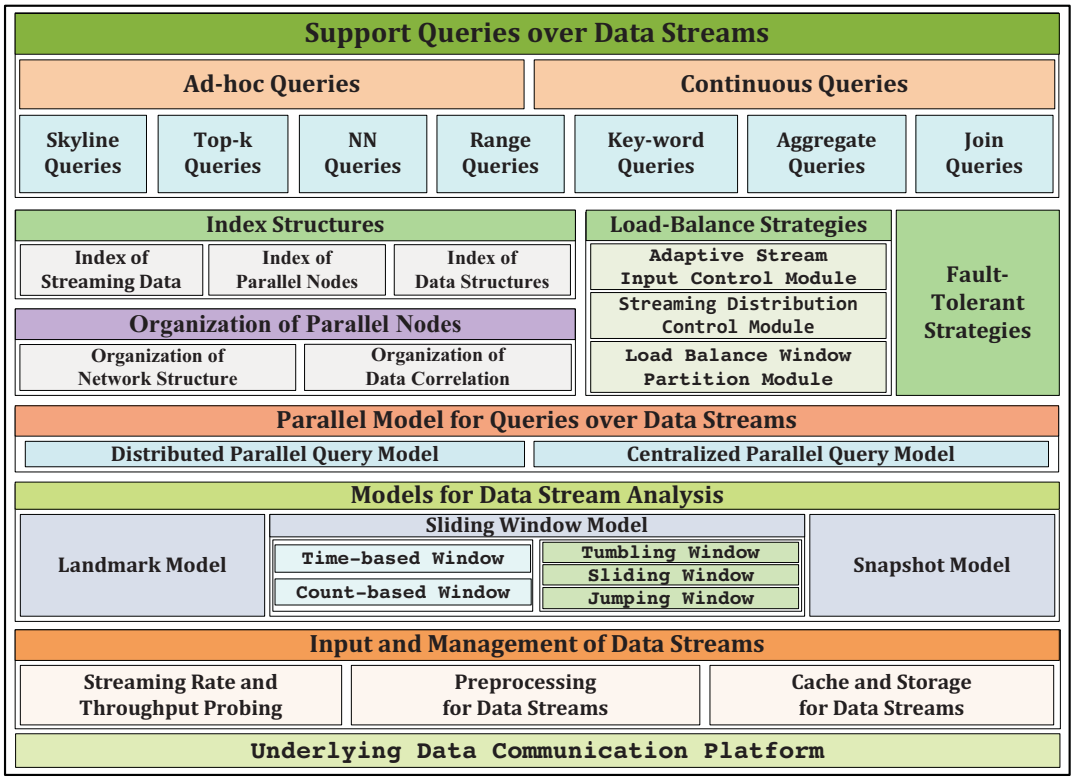


Fig. 1: The architecture of GPS

when we assign tasks based on the window partitioning. Besides, we can dynamically adjust the loads in each node. Thus, exploring dynamic load balancing algorithms or load shedding techniques, is also a very important aspect for designing the general parallel query framework.

3) Integration of Indexing Structures: A large number of studies indicate that the index structure can effectively improve the query processing [8], [18]. We also believe that the index can improve the parallel queries over data streams. First, starting from the global data or the nodes, develop the distributed or parallel index for the whole streaming items or the parallel nodes. Second, study the index techniques to optimize the processing in each node, so as to improve the performance of the whole system. Third, since we may adopt some special data structures to optimize the queries, if we can use some index structures to organize or index the data structures, the query can also be significantly improved.

4) Organization of Nodes and Data: The organization of the processing nodes not only refers to the architecture of the parallel query processing, but also relates to the query efficiency. Generally, the organization of nodes includes the organization of the network connection structure and the organization of the nodes according to the correlation of the streaming items. The former is the fundamental structure for parallel query processing, and the latter is an effective way to optimize the parallel query processing. Similarly, we can also organize the streaming items with a certain data mapping strategies to disseminate the data to the processing nodes to improve the queries. Since the query needs fast response and all the data are maintained in the memory during the query processing, we believe that replication method is more suitable than others like data coding. Therefore, we choose replication method to achieve the fault-tolerance in GPS.

5) Fault-Tolerant Query Processing: Failures are normal rather than exceptional in the cloud computing environments. For example, Google has reported 5 average worker deaths per Map-Reduce job [12], and at least one disk failure in every run of a 6-hour Map-Reduce job on a cluster of 4,000 machines [19]. Moreover, as the number of components increases, so does the probability of failure. Once failures are happened during the query process, the query results would be incorrect and the continuous query may be interrupted, which waste plenty of computing resources and seriously affects the query experience of the users. Therefore, we need to study the fault-tolerant strategies to solve the problems, which include the fault detection, task migration and rapid recovery for the queries. Thus, we need to recover the parallel queries with minimal cost, such as to make full use of the results that have already calculated and avoid the problem of double counting.

6) Evaluation Methods and Metrics: To evaluate the performance of a parallel framework for the queries over data streams, a variety of evaluation approaches or metrics must be considered in its design. The approaches or the metrics should have a certain versatility and fairness, as well as can

practically reflect the performance of parallel processing. The typical metrics for evaluation in GPS include: query processing rate, load-balance, scalability, fault-tolerant or robustness, the query complexity in time and space.

III. General Parallel Model

A. Structure of Parallel Model

In this section, we propose a hierarchical and general parallel model for queries over data streams, which corresponds to the module for parallel processing in our proposed framework. The structure of the model is shown in Fig. 2.

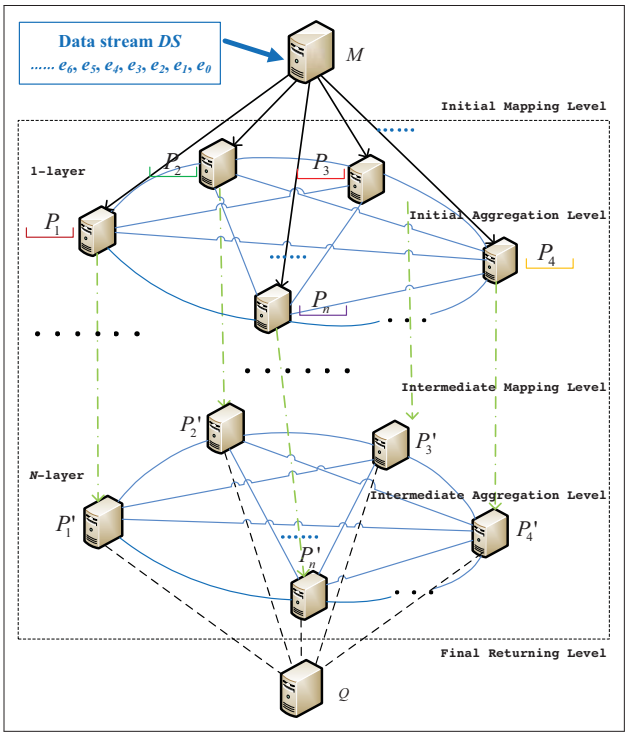


Fig. 2: The hierarchical and general parallel processing model for queries over data streams

Generally, there are two ways to translate a centralized processing to parallel processing, i.e., data partitioning and task partitioning, which are fully considered in our parallel model. First, we parallelize the query by partitioning the whole query tasks into many subtasks and assign each task to a layer, thus the query processing can be vertically parallelized. Second, we further parallelize the query by assigning the processing tasks on the whole streaming data in a layer to many processing nodes, thus all the nodes in a layer are involved in the parallel processing for the tasks on their own local streaming items.

As shown in Fig. 2, the whole general parallel processing model is organized as a hierarchical parallel processing structure. The model includes several layers, and the number of layers is defined by the users. In each layer, we organize the nodes to parallel execute the tasks assigned in the layer. The nodes in the upper layers connect with the nodes in the lower layers with some mapping rule, and the nodes in each layer can in parallel execute the tasks for the queries.

Generally, all the parallel queries over data streams based on the aforementioned parallel model mainly include the following five processing levels.

- *Initial mapping level*: we firstly map the active streaming data to the nodes in the first layer with a certain mapping rule defined by the users.
- *Initial aggregation level*: the nodes in this layer execute the query processing in parallel in the first layer, then push the results to the next level.
- *Intermediate mapping level*: we map the results produced by the upper level with a certain rule to initiate a new round of parallel processing in the layer.
- *Intermediate aggregation level*: similar to the initial aggregation level, the main task in this layer is to further execute the task that assigned in it.
- *Final returning level*: this layer is mainly responsible for collecting the final results from the calculation by the final processing layer.

Generally, in many queries over data streams, we may only need to consider for the processing with one layer, but in some complex queries, the query processing may refer to several layers, thus to do the parallel query processing with our proposed framework can significantly improve the queries. In particular, the nodes between two adjacent layers not necessarily with one by one mapping rule, and we only need to design the rules for the specific queries.

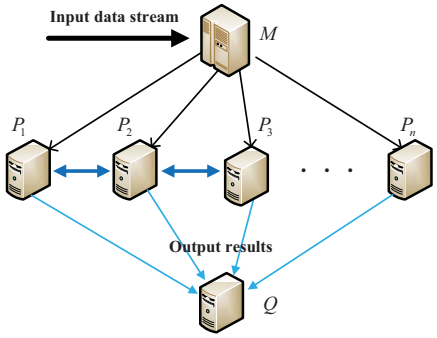


Fig. 3: The parallel model with 1-layer

For simplicity, we abstract one layer to explain the general query processing with the proposed parallel model. Specifically, as shown in Fig. 3, there exist three kinds of nodes in our parallel processing settings:

- *Monitor node* (M): the node is responsible for mapping and delivering the arriving streaming items to the parallel computing nodes;
- *Peer node* ($P_i\,(1 \leq i \leq n)$): the nodes are responsible for the computation for their maintained local streaming items, which can communicate with M;

- *Query node* (Q): the node is responsible for outputting the final updated query results from all the processing nodes to the users.

Note that, in each layer there actually exist two levels, i.e., a mapping level and an aggregation level. Although this structure is similar to the MapReduce, GPS is more flexible than MapReduce, as all the nodes can directly communicate with others in each layer. Besides, many assistant nodes can also be used in each layer to improve the query if necessary, thus the role of each node can be flexibly defined and not only the map or reduce node.

B. Parallel Model in Each Layer

In the aforementioned general model for parallel query processing over data streams, each layer is individually designed for different query tasks. In our design of each layer, two essential types of parallel processing models are proposed: centralized parallel model (CPM) and distributed parallel model (DPM).

1) Centralized Parallel Model: As shown in Fig. 4, M can directly communicate with the peer nodes, but the peer nodes need not communicate with each other. When the streaming data comes to node M, M forwards the data to the peer nodes with some user defined rule. Each peer node maintains two sliding windows W and W_i, where W denotes the global sliding window and W_i is the local sliding window. Note that, each peer node is responsible for processing the queries for the data in the window W_i. In this model, the global sliding window W are divided into multiple sub-windows $W_1, W_2, \ldots, W_n$, which are assigned to the nodes $P_1, P_2, \ldots, P_n$, respectively. Specifically, it satisfies that $W = \bigcup_{i=1}^{n} W_i$ and $W_i \cap W_j = \emptyset\,(i \neq j)$, which means that there is no overlap between any two sub-windows.

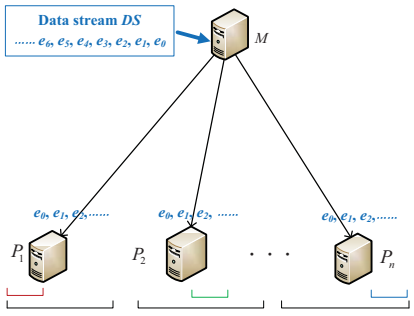


Fig. 4: Centralized parallel model

In this model, since each peer node maintains the global skyline window, each node can ensure the correctness of the querying results and need not communicate with each other. However, since this model demands that each node maintains two sliding windows in memory and the size of the sliding window can be too large, this model may consume too much memory resources and decrease the processing efficiency when the memory space is not enough. From the analysis, we can see that this parallel model try to exchange the communicate overhead with the memory overhead, which is more suitable for the applications that are more concerned about the network communication overhead.

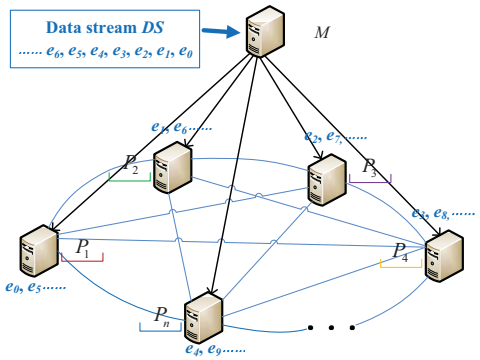


Fig. 5: Distributed parallel model

2) Distributed Parallel Model: To further explore the parallelism and reduce the memory overhead, we propose a distributed parallel model for the queries over data streams. As shown in Fig. 5, M can directly interact with the peer nodes, and all the peer nodes can also communicate with others. When the data arrives to M, M alternatively forwards the data to the peer nodes in the corresponding layer. Each peer node maintains a local sliding window W_i. Similar to CPM, it also satisfies that $W = \bigcup_{i=1}^{n} W_i$ and $W_i \cap W_j = \emptyset\,(i \neq j)$ in DPM.

In DPM each peer node does not need to maintain any global sliding window, but only a local sliding window is maintained. Peer nodes can execute the parallel queries by communicating with others when the interactions for the queries are needed. Thus, DPM can efficiently realize the parallel query processing with less memory overhead.

Note that, in both CPM and DPM another important problem that we need to address is the load balance. Particularly, in order to achieve the target, we try to assign the partitions of the global sliding window to the peer nodes with different sizes according to the capabilities of the peer nodes in logical. Actually, the capability may be determined by the processing capability, storage capability and network bandwidth. Since each node may run a variety of services in cloud, the remaining resources of each node may constantly change at different times. Thus, to measure the comprehensive capability of each peer node, we try to run the query over multiple sampled uncertain data sets with different scales to evaluate the capability of each node, in order to achieve the load balancing purpose.

As shown in Algorithm 1, we first sample some uncertain datasets with different sizes to test the processing capability of each peer node, which is measured by the processing rate (Line 4), and then get the the average rates by computing the average values for several test times (Line 5). Afterwards, we can further obtain the overall average processing rates with Line 7, and thus we assign the sizes of the windows according to the overall capability weights with Lines 8∼9.

Algorithm 1: Load balance oriented window partitioning

Input : The length of global sliding window $|W|$; the number of peer nodes n;
Output: The size of each sub-window $|W_i|\,(1 \le i \le n)$

begin
  Randomly generate m data sets with different scales: $S_1, S_2, \cdots, S_m$;
  Each peer node tests the times $(t_1, t_2, \cdots, t_m)$ of processing the queries on these data sets;
  Each peer node obtains the processing rates with $v_i = |S_i|/t_i, (i = 1, 2, \cdots, m)$;
  Each peer node continuously samples and computes it with l times, and gets the average rates as $\overline{v}_i = (v_{i1} + v_{i2} + \cdots + v_{il})/l, (i = 1, 2, \cdots, m)$;
  foreach *peer node* $i\,(1 \le i \le n)$ **do**
    Compute the overall average processing rate $v'_i = (\overline{v}_1 + \overline{v}_2 + \cdots + \overline{v}_n)/n$;
    Calculate the overall capability weight $w_i = (v'_1 + v'_2 + \cdots + v'_n)/n$;
    Obtain the corresponding size of the sub-window as $|W_i| = w_i \cdot |W|$;
end

C. Example of Uncertain Skyline Queries

In this subsection, we will provide an example of skyline queries over uncertain streams to explain the general parallel model for the parallel query processing.

Assume that we model the data streams with *count-based* sliding windows, and the tuples are processed in a *first-in-first-out* (*FIFO*) manner. Furthermore, suppose the data stream is denoted by DS and the set of most recent N elements in DS is represented by DS_N, which is included in the sliding window W. First, we provide some definitions related to the skyline queries over uncertain streams.

Definition 1: **(dominance)** For any two objects u and v from the d-dimensional space, u *dominates* v, denoted by $u \prec v$, iff $u.i \le v.i$ for all dimensions $1 \le i \le d$ and there at least exists a dimension j satisfies $u.j < v.j$.

Assume that $SKY(W)$ denotes the set of elements in W that form the skyline of W. The probability that an streaming item e appears in the skylines of the possible worlds is $P_{sky}(e) = \sum_{e \in SKY(W), W \in \Omega} P(W)$ [18]. Thus, the skyline probability of e against the data stream DS_N can be defined as follows [5], [8].

Definition 2: **(skyline probability)** Suppose $P(e)$ denotes the existence probability of an item e, then the skyline probability of e over DS_N is defined by

$$P_{sky}(e) = P(e) \times \prod_{a \in DS_N, a \prec e} (1 - P(a)) \qquad (1)$$

Based on the above definition, we can define the q-skylines in DS_N that is represented by $SKY_{N,q}$ as follows:

Definition 3: **(q-skyline in DS_N)** Given a probability threshold $q \in (0, 1]$, q-skyline in DS_N is defined by a subset of DS_N, where the skyline probability of each streaming item in the subset is not less than q. That is, for each element e in $SKY_{N,q}$, we have $P_{sky}(e) \ge q$.

Problem Definition: Given an uncertain data stream DS_N containing N active items $e_1, e_2, \ldots, e_N$ in a sliding window DS_N, where N is the window size, query the q-skyline items continuously in DS_N.

Take queries with DPM for example, M first transmits e_0 to P_1, and then transmits e_1 to P_2, followed by e_2 to P_3, and so forth. Specifically, if the window of one peer node P_i is full and it receives a new item e_{new}, then it will transmit the new item and the expired item e_{old} to all the other peer nodes $P_j\,(1 \le j \le n, j \ne i)$. After receiving the items from P_i, P_j computes the dominating probability $P_d^j(e_{new})$ of e_{new} against the local window W_j with $P_d^j(e_{new}) = \prod_{e_i \in W_j, e_i \prec e_{new}} (1 - P(e_i))$, and then sends the value to P_i. Meanwhile, P_j updates the skyline probabilities for all its maintained items in W_j and delivers the updated skylines to node Q. Thus, when P_i receives the $P_d^j(e_{new})\,(1 \le j \le n, j \ne i)$ values from all the other peer nodes, it can compute the global skyline probability of e_{new} with $P_{sky}(e_{new}) = P(e_{new}) \times \prod_{j=1}^{n} P_d^j(e_{new})$.

IV. Empirical Evaluation

In order to verify the effectiveness and efficiency of the proposals, we have implemented the prototype of the general framework and taken uncertain skyline queries for an example to test the performance of the queries with it.

We conduct the experiments with real deployment on a small-sized data center, which includes 48 hosts within 3 clusters. All the hosts are homogeneous, each of which is configured with a dual-core Intel 2.6GHz Xeon CPU, 2G main memory, a 1TB hard disk and gigabit ethernet. In addition, each host is deployed with two virtual machines, each of which corresponds to a peer node in the parallel model. The framework is implemented with Java running on the CentOS operating system. We run each experiment five times and get the average value.

Moreover, we use synthetic data [20] for the experiments, and use Normal distribution to randomly assign an existence probability to each streaming tuple with mean value $\mu = 0.7$ and standard deviation $\sigma = 0.3$. Specifically, assume that the layer of the framework is 1 and adopt the CPM and DPM to process the query. Meanwhile, suppose that the default size of the global sliding window is 2×10^6, and the default dimensionality of the streaming tuple is 4.

A. Performance of Load Balancing

To analyze the load balance of the proposed algorithm, we use 10 peer nodes to carry out the skyline computation and differ the capabilities of the nodes by allocating different resources to each virtual machine. According to the Algorithm 1, we assign the size of each local sliding window based on the capability of the peer nodes, which is measured by the overall processing rate for the queries on some sampled data sets.

In this experiment, we have sampled three Independent datasets with sizes of 1M, 10M and 100M, and then we can draw the overall processing rate to get the overall capability

TABLE I: Load Balance Evaluation

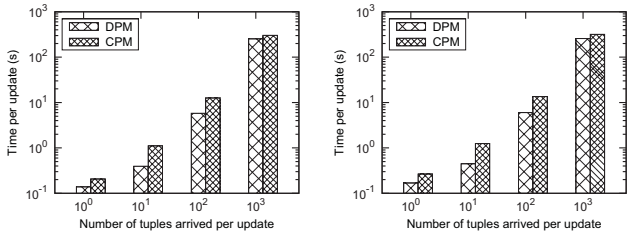
Peer node ID	1	2	3	4	5	6	7	8	9	10
Capability weight	0.022	0.041	0.062	0.083	0.096	0.105	0.115	0.134	0.163	0.179
Local window size (M)	0.11	0.205	0.31	0.415	0.48	0.525	0.575	0.67	0.815	0.895
Time per update (DPM)	11.11	11.91	11.42	12.03	11.57	11.78	12.34	11.45	11.09	11.89
Time per update (CPM)	17.32	18.03	17.89	17.37	17.44	18.39	18.52	17.61	17.25	17.99

weight, thus we can get the size of the corresponding local sliding window. The Independent streaming data with dimensionality $d = 4$, size of global sliding window $|W| = 5M$, update granularity $m = 10^2$, and number of peer nodes $n = 10$ are used in this experiment.

Table I gives the exclusive cpu time per update for updating the skyline computation of the 10 peer nodes. As can be seen from the table, the proposed algorithm can achieve good load balance among all the peer nodes in CPM and DPM. In Algorithm 1, the monitor node M divides the data fairly according to the capabilities of the peer nodes, which contributes to the ideal query load balancing.

B. Performance with Update Granularity

Since the streaming data is continuously arrived, and sometimes a batch of items may arrive at a time, thus we need to evaluate the performance with the number m of the arrivals at a time. To evaluate the effect of the numbers, in our experiments we vary m from 1 to 10^3. As shown in Fig. 6, we can see that the time per update for two models increases as m increases for both the synthetic and real data, due to the time-consuming for the query updating for the larger number of arrivals. Nevertheless, we also find that the time is not linearly increasing as the increase of m. This is probably a consequence of that, when m is small, the communication overhead increases due to the frequent data transmission, whereas when m is large, the vast number of dominating tests increase the computation overhead.



(a) Independent data (b) Anti-correlated data

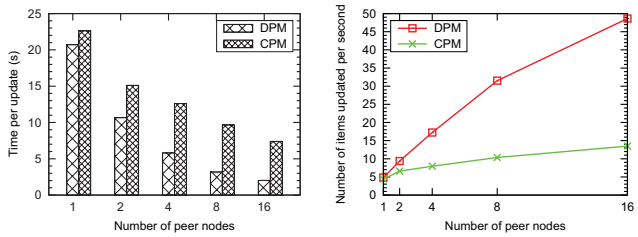
Fig. 6: Performance *vs.* Update granularity m

Furthermore, as shown in Fig. 6, we can find that the performance of DPM is better than CPM, which stems from that the computation of the skyline probabilities for the new arrivals is only occupied by a peer node in CPM, while in DPM the tasks are executed by all peer nodes. Thus, the computation is improved, especially in the cloud environment. To be more obviously evaluate the models, we choose 100 as the default value for the remaining tests.

Moreover, from Fig. 6 we can also see that the performances of CPM and DPM in two types of synthetic data are close, this is probably the consequence of that we compute the dominating probabilities for new items and update the skyline probabilities based on the complete dominance tests in the models. Therefore, in the following experiments we only use Independent data to evaluate our proposals.

C. Performance with Peer Number

To evaluate the scalability of the models, we test the performances of the models with different number n of peer nodes. From Fig. 7(a), we can see that the time per update for continuous uncertain skyline queries in each model is reduced when the number of peer nodes increases from 1 to 16. The results confirm that the more peer nodes are adopted to parallelize the query, the more improvement will be, especially for the query over the large sliding window. Therefore, we believe that the proposed models are scalable.



(a) Update time *vs.* n (b) Update rate *vs.* n

Fig. 7: Performance *vs.* Peer number n

Moreover, from Fig. 7(b) we can see that the performance which is measured by the number of items updated per second is improved as the increase of the number of nodes. Nevertheless, the growths are relatively slowly compared with the growth rates of peer numbers. The main reason is that the size of each local window decreases gradually with the increase node number, thus the computational overhead on each node decreases, whereas the communication overhead increases, which affects the performance of the models.

D. Performance with Size of Global Sliding Window

To evaluate the adaptivity of the framework for the size of the global sliding window $|W|$, we vary $|W|$ from 0.5×10^6 to 5×10^6. Moreover, we assign the global window to n local windows with equal sizes $|W_i| = |W|/n\,(1 \leq i \leq n)$ to achieve the goal of load balance as far as possible.

From the experiments, we find that framework can effectively process the queries even when the sliding window is large. As shown in Fig. 8, the overall time per update increases when we vary the size of global window from $0.1M$ to $5M$. This is nature as that the larger size of the global window

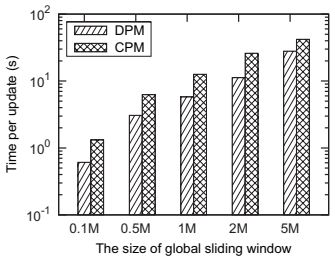


Fig. 8: Performance *vs.* Window size

is, the more recent stream items that need to be maintained will be, thus the computation overhead including the cost for updating the existing items and the cost for computing the skyline probabilities for new arrivals increases. Furthermore, as it is expected the performance of DPM is better than CPM, due to the more parallelism for computing the skyline probabilities of the new arrivals.

V. Conclusion

Nowadays, parallel query processing over data streams has received considerable attention in academic and commercial areas, but it poses an immense challenge to researchers. In this paper, we present a general framework GPS for parallel queries over data streams for distributed computing environments with high performance [21], [22], especially for the cloud computing environments. To the best of our knowledge, it is the first work to study the problem. Extensive experimental results demonstrate that GPS is effective and practical.

As future work, it is interesting and challenging to improve the framework in several aspects, such as the index techniques and interface standardization of the framework. Moreover, we will test the performance of the framework for other queries, such as uncertain top-k queries and the queries over the graphical streaming data. Additionally, we are currently studying the fault-tolerant techniques for parallel query processing over uncertain data streams based on the proposed framework GPS.

Acknowledgement

This work was supported by the National Grand Fundamental Research 973 Program of China (Grant No.2011CB302601), the National Natural Science Foundation of China (Grant No.61379052), the National High Technology Research and Development 863 Program of China (Grant No.2013AA01A213), the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (Grant No.S2010J5050), Specialized Research Fund for the Doctoral Program of Higher Education (Grant No.20124307110015).

References

[1] C. Aggarwal and P. Yu, "On indexing high dimensional data with uncertainty," in *Proceedings of the 24th International Conference on Data Engineering (ICDE)*. IEEE, 2008.

[2] Y. Wang, X. Li, X. Li, and Y. Wang, "A survey of queries over uncertain data," *Knowledge and Information Systems (KAIS), DOI: 10.1007/s10115-013-0638-6*, 2013.

[3] J. Kurose, E. Lyons, D. McLaughlin, D. Pepyne, B. Philips, D. Westbrook, and M. Zink, "An end-user-responsive sensor network architecture for hazardous weather detection, prediction and response," *Technologies for Advanced Heterogeneous Networks II*, pp. 1–15, 2006.

[4] Y. Diao, B. Li, A. Liu, L. Peng, C. Sutton, T. Tran, and M. Zink, "Capturing data uncertainty in high-volume stream processing," in *Conference on Innovative Data Systems Research (CIDR)*, 2009.

[5] Y. Yang and Y. Wang, "Towards estimating expected sizes of probabilistic skylines," *Science China: Information Sciences*, vol. 54, no. 12, pp. 2554–2564, 2011.

[6] R. Cheng, L. Chen, J. Chen, and X. Xie, "Evaluating probability threshold k-nearest-neighbor queries over uncertain data," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT)*. ACM, 2009, pp. 672–683.

[7] X. Lian and L. Chen, "Monochromatic and bichromatic reverse skyline search over uncertain data," in *Proceedings of the international conference on Management of data (SIGMOD)*. ACM, 2008, pp. 213–226.

[8] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. Yu, "Probabilistic skyline operator over sliding windows," in *Proceedings of the 25th International Conference on Data Engineering (ICDE)*. IEEE, 2009, pp. 1060–1071.

[9] D. Sun, G. Chang, S. Gao, L. Jin, and X. Wang, "Modeling a dynamic data replication strategy to increase system availability in cloud computing environments," *Journal of computer science and technology (JCST)*, vol. 27, no. 2, pp. 256–272, 2012.

[10] H. Köhler, J. Yang, and X. Zhou, "Efficient parallel skyline processing using hyperplane projections," in *Proceedings of the international conference on Management of data (SIGMOD)*. ACM, 2011, pp. 85–96.

[11] A. Safaeu and M. Haghjoo, "Parallel processing of continuous queries over data streams," *Distributed and Parallel Databases*, vol. 28, no. 2-3, pp. 93–118, 2010.

[12] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the Conference on Operating System Design and Implementation (OSDI)*, 2004, pp. 137–150.

[13] T. Condie, N. Conway, P. Alvaro, J. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. USENIX Association, 2010, pp. 21–21.

[14] J. Ekanayake, S. Pallickara, and G. Fox, "Mapreduce for data intensive scientific analyses," in *Fourth IEEE International Conference on eScience*. IEEE, 2008, pp. 277–284.

[15] Y. Bu, B. Howe, M. Balazinska, and M. Ernst, "Haloop: Efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.

[16] J. Dittrich, J. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing)," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 515–529, 2010.

[17] S. Muthukrishnan, *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[18] X. Ding and H. Jin, "Efficient and progressive algorithms for distributed skyline queries over uncertain data," *IEEE TKDE*, vol. 24, no. 8, pp. 1448–1462, 2012.

[19] J. MacCormick, N. Murphy, M. Najork, C. Thekkath, and L. Zhou, "Boxwood: Abstractions as the foundation for storage infrastrucure," in *Proceedings of OSDI*, 2004.

[20] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings of the 17th international conference on data engineering (ICDE)*. Citeseer, 2001, pp. 421–430.

[21] Y. Wang and S. Li, "Research and performance evaluation of data replication technology in distributed storage systems," *Computers & Mathematics with Applications*, vol. 51, no. 11, pp. 1625–1632, 2006.

[22] X. Lu, H. Wang, J. Wang, J. Xu, and D. Li, "Internet-based virtual computing environment: Beyond the data center as a computer," *Future Generation Computer Systems (FGCS)*, vol. 29, no. 1, pp. 309–322, 2013.