Incremental Join Aggregate Algorithms Based On Compound Sliding Window

Dhananjay M.Kanade
Department of Computer Engineering,
KKWIEER,Nasik

Kushal P. Birla
Department of Computer Engineering,
KKWIEER,Nasik

ABSTRACT
Incremental join aggregate queries based on sliding window are a sort of queries that are widely used. All the join aggregate query algorithms in existing research works are designed for immediate continuous queries. A join aggregate query method based on Compound sliding window for periodically executed continuous queries is presented. This method organizes the basic windows in a compound sliding window according to their join properties, the aggregate values are computed while the join processing, the join results of compound sliding window are not saved, so the memory used by query processing is greatly reduced. Some of which we discussed in CJMAX where the input data stream is serial. CJMAX algorithm possesses superior performance. There will be good time and space complexity of this incremental algorithm. Many applications in several domains require online processing of continuous data flows. They produce very high loads that require aggregating the processing Capacity of many nodes. Current Stream Processing Engines do not scale with the input load due to single-node bottlenecks. Here new concept Cloud Stream, a scalable and elastic stream processing engine for processing large data stream volumes. Cloud Stream uses a novel parallelization technique that splits queries into sub queries that are allocated to independent sets of nodes in a way that minimizes the distribution overhead .Its elastic protocols exhibit low intrusiveness, enabling effective adjustment of resources to the incoming load. Elasticity is combined with dynamic load balancing to minimize the computational Resources used.

General Terms
Data stream, join aggregate algorithms

Keywords
Basic windows, compound sliding window, Data stream, join aggregate algorithms

1. INTRODUCTION
1.1. Detail problem definition

In many applications of data stream, join aggregate queries based on sliding window are a sort of queries that are widely used. All the join aggregate query algorithms in existing research works are designed for immediate continuous queries. A join aggregate query method based on compound sliding window for periodically executed continuous queries is presented. This method organizes the basic windows in a compound sliding window according to their join properties, the aggregate values are computed while the join processing, the join results of compound sliding window are not saved, so the memory used by query processing is greatly reduced. An algorithm that computes the N+1th join aggregate value increment by using the Nth one is presented. Theoretical analysis and experiment result both show good time and space complexity of this incremental algorithm

A number of application scenarios where massive amounts of data must be processed in quasi-real-time are showing the limits of the traditional “store-then-process” paradigm. In this context, researchers have proposed a new computing paradigm based on Stream Processing Engines (SPEs). SPEs are computing systems designed to process continuous streams of data with minimal delay. Data streams are not stored but are rather processed on-the-fly using continuous queries. The latter differs from queries in traditional database systems because a continuous query is constantly “standing” over the streaming tuples and results are continuously output

1.2 Justification of problem

As the great development in computer implementation, data stream query processing technique becomes a focus of current data base research. A great deal of research has been done in this field, and many effective data stream query processing techniques are presented. In many applications of data stream, aggregate query of data streams join result is often required; we call this sort of query as “join aggregate query”. Because the data stream is infinite and the limitation of system memory, a join aggregate query can only give an approximate result. Sliding window is a data sampling technique, which is often used in data stream query operation and approximate result computation. A sliding window on a data stream is a window that is set on a segment of the data stream, and the segment only includes the latest coming data in the data stream. When there is new data coming, the sliding window will move ahead, replacing the oldest data with the latest coming fresh one. Comparing with random sampling and other sampling techniques, the approximation meaning of sliding window is more obvious. It is more important that, in many applications, the user is most interested in the latest coming data in the data stream. So, sliding window is an ideal data sampling technique for data stream query operations.

1.3 Need of proposed system Data stream queries are mainly continuous queries, the query algorithm continuously returns query results as the new data come continuously. There are two execution manners of data stream continuous queries: one is immediate execution manner; and the other is periodical execution manner. For immediate execution manner, an execution of query is triggered every time when there is a new data coming in the data stream; and for periodical execution manner, the query operation is executed at fixed time intervals. At present, all the join aggregate query algorithms based on sliding window are designed for immediate continuous queries, but in practical applications, in some cases the query results in a time period are needed. For example, when monitoring the operation of internet, the network operation needs to know and analyze the operation condition of the network between two integral points. So the data that arrives between the two integral points can not be inserted into the sliding window to do query operation and to get query results, until the next integral point. Similar requirements also exist in some other application fields. A compound sliding window two-way join simple aggregate query algorithm for periodically executed continuous query is presented. A compound sliding window consists of several equally sized basic windows. When the latest basic window is full, it will be inserted into the compound sliding window, and expired basic window in the compound sliding window will be deleted. The main idea of this algorithm is to calculate aggregate value while doing join processing; the join result of compound sliding window will not be saved, so the memory required for the query operation is significantly reduced. For join aggregate query algorithm, a method of calculating the N+1th join aggregate value from the Nth join aggregate value is considered. Theoretical analysis and experiment result both show good time and space complexity of this incremental algorithm.

1.4 Applications A data stream system can be found in many application domains including finance, web applications, security, networking, and sensor monitoring. Traderbot is a web-based financial search engine that evaluates queries over real-time streaming financial data such as stock tickers and news feeds. The Traderbot web site gives some examples of one-time and continuous queries that are commonly posed by its customers. Modern security applications often apply sophisticated rules over network packet streams. For example, iPolicy Networks provides an integrated security platform providing services such as firewall support and intrusion detection over multi-gigabit network packet streams. Such a platform needs to perform complex stream processing including URL-filtering based on table lookups, and correlation across multiple network traffic flows. Large web sites monitor web logs (clickstreams) online to enable applications such as personalization, performance monitoring, and load-balancing. Some web sites served by widely distributed webservers (e.g., Yahoo) may need to coordinate many distributed clickstream analyses, e.g., to track heavily accessed web pages as part of their real-time performance monitoring. An overwhelming amount of transaction log data is generated from telephone call records, point-of-sale purchase (e.g., credit card) transactions, and Web server logs . On-line analysis of transaction logs can identify interesting customer spending patterns or possible credit card fraud.

In the networking community, there has been a great deal of recent interest in using a data management system for on-line monitoring and analysis of network. Routing system analysis and customer billing as well as detecting suspicious activity such as equipment malfunctions or denial-of-service attacks. In this context, a data stream is composed of IP packet headers.

There are several emerging applications in the area of sensor monitoring where a large number of sensors are distributed in the physical world and generate streams of data that need to be combined, monitored, and analyzed.

2. CURRENT SURVEY

2.1 Study of existing systems/ technologies Traditional database systems have proven to be well-suited to the organization, storage, and retrieval of finite datasets. In recent years, however, new data-intensive application shave emerged that need to process data which is continuously arriving at the system in the form of potentially unbounded, time-varying sequences of data items, termed data streams. Examples belonging to this new class of stream-oriented applications can be found in a diversity of application domains including sensor monitoring, finance, transaction log analysis, and network security

Sensor networks are used in a variety of applications for monitoring physical or environmental conditions, for instance, in traffic management, location tracking, Supply chain management based on the upcoming RFID technology, medical monitoring, and manufacturing processes. Queries may involve the detection of unusual and often complex conditions, even across multiple streams, with the aim to activate alarms or trigger actions.

Network traffic management involves on line monitoring and analysis of network packets to find out information on traffic flow patterns for routing system analysis, bandwidth usage statistics, and network security. In particular, intrusion detection requires real-time responses, for instance, to prevent denial-of-service attacks. Electronic trading often relies on the online analysis of financial data obtained from stock tickers and news feeds. Specific goals include discovering correlations, identifying trends, and forecasting stock prices. More current results maximize arbitrage profits.

Transaction logging is performed in many applications, generating huge volumes of data, for example, web server logs and click streams, telephone call records, user account logging, or event logs in online auction systems such as eBay. Queries over these streams of log entries could be used to initiate immediate actions on specific customer behavior, detect suspicious access patterns that could indicate fraud or

attacks, or identify performance bottlenecks with the aim to improve service reliability.

2.2 Analysis of existing systems/ technologies A traditional relational database stores a collection of tables, which are inherently unordered and therefore viewed as sets. Data records are relatively static, and are assumed to be valid until explicitly modified or deleted by a user or application. Queries, typically assumed to occur more frequently than data modifications, are executed when posed and their answers reflect the current state of the database. The goal of a database management system (DBMS) is to provide persistent, consistent, and recoverable storage, as well as an efficient query answering mechanism. The relational model has filled the needs of traditional business applications, as evidenced by the commercial success of relational DBMSs. However, the requirements of a number of emerging applications do not t the above description. One particularly interesting change is that data may be generated in real time, taking the form of an unbounded sequence (stream) of values. This shift is being instigated by the following trends and applications

In recent years a dramatic escalation in feed volumes has been observable due to steady technological progress. For example, every day WalMart records 20 million sales transactions, AT&T generates 275 million call records, and eBay logs 10 million bids. Market data feeds can even generate multiple thousands of messages per second. While storing these massive data sets is possible to a certain extent, extracting valuable information from the resultant histories is often extremely expensive because the overwhelming data volumes accumulated over months or even years cannot be searched and analyzed in acceptable time. As a consequence, the archived data is often discarded after the retention time has elapsed, without any prior analysis but to free space for new data. Even in applications

2.3 Comparison of existing systems with proposed system At present, all the join aggregate query algorithms based on sliding window are designed for immediate continuous queries, but in practical applications, in some cases the query results in a time period are needed. For example, when monitoring the operation of Internet, the network operation needs to know and analyze the operation condition of the network between two integral points. So the data that arrives between the two integral points can not be inserted into the sliding window to do query operation and to get query results, until the next integral point. Similar requirements also exist in some other application fields a compound sliding window two-way join simple aggregate query algorithm for periodically executed continuous query is presented. A compound sliding window consists of several equally sized basic windows. When the latest basic window is full, it will be inserted into the compound sliding window, and expired basic window in the compound sliding window will be deleted. The main idea of this algorithm is to calculate aggregate value while doing join processing; the join result of compound sliding window will not be saved, so the memory required for the query operation is significantly reduced. For join aggregate query algorithm, a method of calculating the N+1th join aggregate value from the Nth join aggregate value is considered. Theoretical analysis and experiment result both show good time and space complexity of this incremental algorithm.

3. Technical Details

3.1 Concept

3.1.1 STRUCTURE OF THE COMPOUND SLIDING WINDOW

There are two ways to define a sliding window in data stream: one is to define the sliding window based on sequence order; and the other is to define the sliding window based on time. In this paper, we will investigate the sliding window based on time; the related definitions are as follows. Definition 1 (Data Stream) A data stream is an infinite time sequence that is arranged in an ascending time order S={<s1,t1>,<s2,t2>, …,<si,ti>, …}, where si is the sequence element that appears at time ti.

Definition 2 (Sliding Window) Let T be a time interval, and t>T is a moment of change. We call S[t-T: t] a sliding window of S with a time interval of T, where t and T have same units, and t is the time delay with respect to the starting point of S.

Definition 3 (Basic Window) Let BT be a time interval. t1, t2 are changing time points, let t1-t2=BT. We call S[t1: t2] a basic window of S with time interval BT, where t1, t2 and BT have same units, and t1, t2 are the time delays with respect to the starting point of S.

Definition 4 (Compound Sliding Window) Let S[t-ST: t] be a sliding window of data stream S at time t, and BT is the time interval of a basic window, let ST=nBT. If S[t-ST: t] only changes at the end at each time interval BT, and at the end of the kth time interval of BT it changes to S[t+kBT-ST: t+kBT], then we call S[t-ST: t] a compound sliding window, noted as SBT[t-ST: t]. Where t, ST and BT have same units.

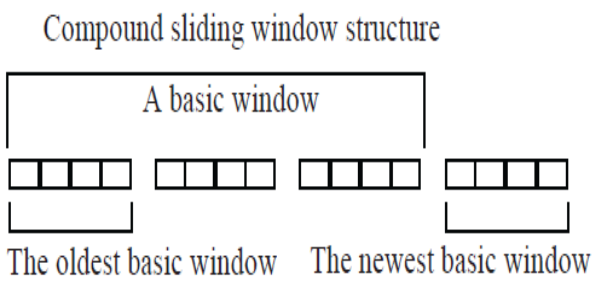

Figure 1. Compound Sliding Window Structure

3.1.2 The Data Stream Model In the data stream model, some or all of the input data that are to be operated on are not available for random access from disk or memory, but rather arrive as one or more continuous data streams. Data streams

differ from the conventional stored relation model in several ways: The data elements in the stream arrive online. The system has no control over the order in which data elements arrive to be processed, either within a data stream or across data streams. Data streams are potentially unbounded in size. Once an element from a data stream has been processed it is discarded or archived it cannot be retrieved easily unless it is explicitly stored in memory, which typically is small relative to the size of the data streams. Operating in the data stream model does not preclude the presence of some data in conventional stored relations. Often, data stream queries may perform joins between data streams and stored relational data. For the purposes of this paper, we will assume that if stored relations are used, their contents remain static.Thus, we preclude any potential transaction-processing issues that might arise from the presence of updates to stored relations that occur concurrently with data stream processing.

3.1.3 Queries

Queries over continuous data streams have much in common with queries in a traditional database management system. However, there are two important distinctions peculiar to the data stream model. The first distinction is between one-time queries and continuous queries . One-time queries (a class that includes traditional DBMS queries) are queries that are evaluated once over a point-in-time snapshot of the data set, with the answer returned to the user. Continuous queries, on the other hand, are evaluated continuously as data streams continue to arrive. Continuous queries are the more interesting class of data stream queries, and it is to them that we will devote most of our attention. The answer to a continuous query is produced over time, always reflecting the stream data seen so far. Continuous query answers may be stored and updated as new data arrives, or they may be produced as data streams themselves. Sometimes one or the other mode is preferred. For example, aggregation queries may involve frequent changes to answer tuples, dictating the stored approach, while join queries are monotonic and may produce rapid, unbounded answers, dictating the stream approach. The second distinction is between predefined queries and ad hoc queries. A predefined query is one that is supplied to the data stream management system before any relevant data has arrived. Predefined queries are generally continuous queries, although scheduled one-time queries can also be predefined. Ad hoc queries, on the other hand, are issued online after the data streams have already begun. Ad hoc queries can be either one-time queries or continuous queries. Ad hoc queries complicate the design of a data stream management system, both because they are not known in advance for the purposes of query optimization, identification of common sub expressions across queries, etc., and more importantly because the correct answer to an ad hoc query may require referencing data elements that have already arrived on the data streams .

3.1.4 Queries over Data Streams

Query processing in the data stream model of computation comes with its own unique challenges. Since data streams are potentially unbounded in size, the amount of storage required to compute an exact answer to a data stream query may also grow without bound. While external memory algorithms for handling data sets larger than main memory have been studied, such algorithms are not well suited to data stream applications since they do not support continuous queries and are typically too slow for real-time response. The continuous data stream model is most applicable to problems where timely query responses are important and there are large volumes of data that are being continually produced at a high rate over time. New data is constantly arriving even as the old data is being processed; the amount of computation time per data element must be low, or else the latency of the computation will be too high and the algorithm will not be able to keep pace with the data stream. For this reason, we are interested in algorithms that are able to confine themselves to main memory without accessing disk.

3.2 Algorithms

3.2.1Join aggregate query algorithms based on compound sliding window

Take aggregate query with MAX for example, to introduce join aggregate query algorithm. For the convenience of description, first introduce some symbols, as shown in Table 1. A compound sliding window two-way join simple aggregate query algorithm is presented. This algorithm calculates aggregate value while doing join processing; the join result of compound sliding window will not be saved, so the memory required for the query operation is significantly reduced. Two practical algorithms of this algorithm are presented: JMAX and CJMAX. The major time consumption of join aggregate query algorithm is the time consumption of join operations, among the compound sliding window join query algorithms, Hash algorithm is the one with the highest execution speed, so we use Hash algorithm for the join operation in the join aggregate query algorithm.

Symbol	Definition
Teswr	Size of compound sliding window
Tcsws	Size of compound sliding widows of data stream S
Tbw	Size of basic windows of data stream R
λr	Arriving speed of data stream R
λS	Arriving speed of data stream S
M	Number of Hash Bucket in Compound Sliding window of data stream R,S.
Ch	Cost Of accessing a tuple in hash algorithm

In JMAX algorithm and CJMAX algorithm, since the compound sliding window join aggregate query algorithm is executed periodically with a period of a basic window, we can make all the tuples in a basic window have the same time stamp. We use the time stamp of the first arriving tuple in a basic window as the time stamp for all the other tuples in the whole same basic window. The algorithm uses a global counter as the time stamp of every tuple that enters the system. The data structure of the compound sliding window is a link sequence. The sequence can be divided into Tcsw/Tbw

blocks. Every block is a basic window structure. The data structure of a basic window is a Hash table that is hashed according to the join property. When a new tuple comes, a new node will be allocated, it will be inserted into the Hash table. There is a head node in a basic window structure, the head node contains a time stamp property and two pointers, one pointer points to the tuples in that basic window, and the other to the head node of the next basic window.

3.2.2 *JMAX Algorithm*

The JMAX algorithm is an intuitive algorithm that is used to calculate the MAX value of the compound sliding window join results. Its basic idea is that, when a new tuple comes in data stream R or S, first insert it into the newest basic window; when the newest basic window is full, it will be inserted into the compound sliding window, and the expired tuples of the compound sliding windows in data stream R and S will be deleted; and then join operation will be done on the tuples in the compound sliding windows of data stream R and S, the join results of the two compound sliding windows will be calculated, as well as the maximum value of the join results. The time complexity of calculating the maximum join value of JMAX algorithm. The JMAX algorithm first inserts the newest basic window to the rear of the compound sliding window, the time cost of this step is the time for an address assignment, and we let it be Ch. The algorithm deletes tuples in the expired basic windows of the compound sliding windows of data stream R and S, we here make an assumption that the data streams have constant data rates, so there are $(r \bowtie s)\times Tbw$ tuples to delete, since the compound sliding window is arranged in a time order, the time cost will be $(r \bowtie s)\times Tbw\times Ch$. Let M be the Number of Hash Buckets in data stream R and S respectively, for the compound sliding window of R, there should be rTcswr/M tuples in every bucket by average; in the same way, for the compound sliding window of S, and there should be λsTcsws/M tuples in every bucket by average. Then the time cost of the join operation of the compound sliding window of data stream R and S will be $\times(\lambda rTcswr/M)\times(\lambda sTcsws/M)\times Ch=(\lambda rTcswr\times\lambda sTcsws)/M\times Ch$. So the Time cost of JMAX Algorithm is:

$$Ch+(\lambda r+\lambda s)\times Tbw\times Ch+(\lambda rTcswr\times\lambda sTcsws)/M\times Ch$$

When JMAX is running, only the two compound sliding windows of data streams R and S need to be stored in memory, there is no additional memory required. JMAX algorithm has the advantage of low memory consumption, but for every time the aggregate value is calculated, all the tuples in the two compound sliding windows need to be joining, and so the time cost is very high. Between two executions of JMAX algorithm, most of the tuples in the compound sliding windows are the same, so when JMAX algorithm runs, a lot of computations are repeated. In the following part CJMAX algorithm will be introduced, which is more efficient.

3.2.3 *CJMAX Algorithm*

When the newest basic window is full, first compare the maximum aggregate value of the tuples in the newest basic window with the current maximum value, if it is smaller than the current one, the tuples in the newest basic window don't need to be joint. The query of compound sliding window join maximum value can fall into two cases: one is that the aggregate property equals the join property; and the other is that the aggregate property is different from the join property.

CJMAX uses the current maximum value as the selection predicates on the two data streams, only the tuples in the newest basic window and with aggregate values that are bigger than the current maximum value need to be joint to calculate the maximum value. the selection predicates can only be set on data stream R. When the newest basic window in data stream S is full, it will be directly joint with the tuples in the compound sliding window of data stream R, the maximum aggregate value of the join result will be calculated, it will be compared with the current maximum value, to select the new maximum value. Let be the current maximum value is the aggregate value of join result r s. If at a new refresh moment, e.g. when the newest basic window is full, a tuple in r s is expired, the current maximum value will be expired too. Then, all the tuples in the two compound sliding windows need to be joined, to calculate a new maximum value. In order to make a judgment of whether or not the current maximum value is expired, we assign a time stamp to the maximum value, its value is the time stamp of the first arriving tuple of r and s. If the aggregate value of r is bigger than or equals to the current maximum value, and there are tuples from several basic windows of the compound sliding window of data stream S satisfy the join condition of r, the time stamp of the maximum value should be the time stamp of the latest tuple matched with r in the basic window of the compound sliding window of data stream S, this time stamp defines the period of validity of the maximum value. Let R. t be the aggregate property.

The time complexities of calculating the maximum join value of CJMAX algorithm. The 1st step of the algorithm is to attach the newest basic window of data stream R to the rear of the compound sliding window, the time cost is the time of an address assignment, let it be. Step 2 to 15 is delete all the tuples in the expired basic windows of the compound sliding windows of data stream R and S, we here make an assumption that the data streams have constant data rates, so there are $(\lambda r+\lambda s)\times Tbw$ tuples to delete, since the compound sliding window is arranged in a time order, the time cost of step 2-15 will be $(\lambda r+\lambda s)\times Tbw\times Ch$. Let p be the probability that the current maximum value is expired. When the maximum value is expired, in step 16-25, two compound sliding windows need to be joint, to calculate the maximum value. Let M be the Number of Hash buckets in data stream R and S respectively, for the compound sliding window of R, there should be λrTcswr/M tuples in every bucket by average; in the same way, for the compound sliding window of S, there should be λsTcsws/M tuples in every bucket by average, so

the time cost will be λrTcswr×λsTcsws/M×Ch. Let q be the probability that the maximum aggregate value of the tuples in the newest basic window of data stream R exceeds the current maximum value. Step 26-42 is, when the maximum value isn't expired newest basic window of data stream R exceeds the current maximum value, use the tuple with the maximum aggregate value in the newest basic window to scan the matched Hash bucket in the compound sliding window of data stream S, to calculate the maximum value. The time cost will be (λsTcsws/M)×Ch. Based on the above discussion, we can see that the average time cost of

CJMAX Algorithm is

Ch=(λr+λs)*Tbw*Ch+(P*λTcswr*λsTcsws/M*Ch)

+(1-p)*q*(λsTcsws/M)*Ch

Inputs: The pointer to the newest basic window of R BWRhead The pointer to the head of the sequence of the compound sliding window of R CSWRfront

The pointer to the rear of the sequence of the compound sliding window of R CSWRrear The pointer to the head of the sequence of the compound sliding window of S CSWSfront The pointer to the rear of the sequence of the compound sliding window of S CSWSrear The tuple with the max aggregate value in the newest basic window of R BWMAX The current maximum join value MaxValue

Outputs: Maximum join value MaxValue

Algorithm Description:

```
1 CSWRrear —> next = BWRhead;
2 Temp1 = CSWRfront;
3 WHILE (CurrentTimestamp - (Temp1 —> timestamp)> Tcswr/Tbw)
4 {
5 CSWRfront = Temp1 —> next;
6 Delete every tuple in the Hash table of the expired basic window led by Temp1;
7 Temp 1= CSWRfront;
8   }
9 Temp2 = CSWSfront;
10 WHILE (CurrentTimestamp - (Temp2 —> timestamp) > Tcsws /Tbw)
11    {
12 CSWSfront = Temp2 —> next;
13 Delete every tuple in the Hash table of the expired basic window led by
    Temp2;
14 Temp 2= CSWSfront;
15     }
16 IF ( CurrentTimestamp- MaxValue.timestamp ≥ Tcswr /Tbw )
17 {
18 FOR (Every Hash Bucket A in every basic window led by CSWRfront)
19  {
20 FOR (Every Bucket B that has the same Hash value as A, for every Hash Bucket A in every basic window led by CSWRfront)
21   {
22 Join the tuples with the same join property in Bucket A and B, and calculate MaxValue
23    }
24    }
25   }
26 ELSE IF (BWMAX.t ≥ MaxValue) //t is the aggregate property
27 {
28 FOR (Every basic window led by CSWSfront)
29 {
30 Search for the Bucket B with the same Hash value with BWMAX
31 FOR (Every tuple s in Bucket B)
32   {
33 IF (BWMAX.l == s.l) // l is the join property
34 IF (BWMAX.t > MaxValue)
35 MaxValue = BWMAX.t;
36 MaxValue.timestamp = s.timestamp;
37 ELSE IF (s.timestamp > MaxValue.timestamp)
38 MaxValue.timestamp = s.timestamp;
39       }
40      }
41   }
42 Output MaxValue;
```

3.3 Performance Analysis

Because CJMAX algorithm don't need to scan the entire compound sliding window every time, its maximum value can't be expired in the first 100 seconds, the time cost of this algorithm is relatively low. After the first 100 seconds, since the maximum value may become expired, it may have to scan part of the compound sliding window, so the time cost of the algorithm will grow, but it will never exceed the time cost of JMAX algorithm. If there are quite a lot of tuples in a basic window, the maximum value will seldom become expired. CJMAX algorithm possesses superior performance over JMAX algorithm. When the aggregate value of the tuples in data stream R is ordered, the chance that the maximum value becomes expired increased, the overall performance of CJMAX algorithm will decrease, but it still won't be lower than the performance of JMAX algorithm .

4. Future Work

4.1 Cloud Stream

In the last few years, there have been substantial advancements in the field of data stream processing. From centralized SPEs(Stream Processing Engines)], the state of the art has advanced to SPEs able to distribute different queries among a cluster of nodes (inter-query parallelism) or even distributing different operators of a query across different nodes (inter-operator parallelism). However, some applications have reached the limits of current distributed data streaming infrastructures. For instance, in cellular telephony, the number of Call Description Records (CDRs) that must be

processed to detect fraud in real-time is in the range of 10,000-50,000 CDR/second. In such applications, most queries for fraud detection include one or more self-joins of the CDR stream using complex predicates, requiring comparison of millions of CDR pairs per second.

Such applications call for more scalable SPEs that should be able to aggregate the computing power of hundreds of cores to process millions of tuples per second. The solution to attain higher scalability and avoid single-node bottlenecks of current SPEs, lies in architecting a parallel-distributed SPE with intraoperator parallelism. However, this requires addressing a number of additional challenges. Parallelization should be syntactically and semantically transparent. Syntactic transparency means that query parallelization should be oblivious to the user. That is, users write a regular (i.e., non-parallel) query that is automatically parallelized by the system. Semantic transparency means that, for a given input, parallel queries should produce exactly the same output as their non-parallel counterparts. On the other hand, resource usage should be cost effective. Many applications exhibit sudden, dramatic changes in the workload that can result in a variation of 1-2 orders of magnitude between peak and valley loads. For instance, in Idea cellular phone networks, streams of CDRs reach peaks of hundreds of thousands of records, while valley loads are in the range of thousands of records per second. A parallel but static SPE, that is, deployed on a fixed number of processing nodes, leads to either under-provisioning (i.e., the number of nodes cannot handle the workload) or over-provisioning (i.e., the allocated nodes are running below their full capacity).Under-provisioning results in the violation of service level agreements that can incur high economic costs. Even with best effort agreements, under-provisioning is responsible for losing unhappy users and raising bad reputation from disgruntled users. Over-provisioning is not cost-effective as resources are not fully utilized. A parallel SPE should be elastic and adjust the amount of its resources (i.e., the number of allocated nodes) to the current workload. Moreover, elasticity should be combined with dynamic load balancing. Without dynamic load balancing, the system would provision new nodes as a result of uneven load distribution. Therefore, the saturation of a single node would lead to unnecessary provisioning of new instances. With dynamic load balancing, new nodes are provisioned only when the system as a whole does not have enough capacity to cope with the incoming load

Here it present Cloud Stream and provides transparent query parallelization. That is, users express regular queries that are automatically parallelized. A compiler takes the abstract query and generates its parallel version that is deployed on a cluster of nodes. The system features high scalability, elasticity and load balancing. SC provides high scalability by exploiting intra-operator parallelism. That is, it can distribute any subset of the query operators to a large set of shared-nothing nodes. Logical data streams are split into multiple physical data sub streams that flow in parallel, thus avoiding Single-node bottlenecks .Communication across different nodes is minimized and only performed to guarantee semantic transparency. SC performs content-aware stream splitting and encapsulates the parallelization logic within smart operators that make sure that the outcome of the parallel execution matches the output of the centralized execution. SC monitors its activity and dynamically reacts to workload variations by re-organizing the load among its nodes as well as provisioning or decommissioning nodes. Elastic resource management is performed on-the-fly and shows very low intrusiveness, thus making provisioning and decommissioning cost-effective. The contributions can be summarized as follows:

1. A highly scalable and elastic SPE for shared-nothing clusters. SC is a full-fledged system, with a complete implementation currently being used for industrial applications.

2. A novel parallelization approach that minimizes the distribution overhead.

3. Transparent parallelization of queries via a query compiler.

4. Effective algorithms for elastic resource management and

Load balancing that exhibit low overhead;

5. CONCLUSION

A compound sliding window join aggregate query algorithm that uses basic windows as refresh period is presented. Compound sliding window join aggregate query algorithms calculate aggregate values while doing join processing, the join results of compound sliding windows don't need to be stored, so the memory consumption of the query operation has been greatly reduced. For join aggregate query algorithm, considerations have been taken to calculate the N+1th join aggregate values using the Nth ones the cost analysis of algorithms are made, and the performances of algorithms are shown by experiments. Also cloud Stream a highly scalable and elastic data streaming system. Cloud Stream provides transparent parallelization that preserves the syntax and semantics of centralized queries. Scalability is attained by means of a novel parallelization strategy that minimizes the distribution overhead. Elasticity and dynamic load balancing minimize the number of resources used for coping with varying workloads.

6. REFERENCES

[1] Zhong Ying-li. "Join Algorithms of Compound Sliding Window"[J].Journal of Harbin University of Commerce; Natural Sciences Edition .2004,20(3):294-299

[2] "Query Processing, Resource Management, and Approximation in a Data Stream Management System" Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu,Mayur Datar, Gurmeet Manku, Chris Olston, Justin Rosenstein, Rohit Varma Stanford University 2003

[3] Motwani R, Widom J, Arasu A, Babcock B, Babu S, Datar M, Manku G, Olston C, Rosenstein J, Varma R. "Query processing, resource management, and approximation in a data stream management system." In: Stonebraker M, Gray J, Dewitt D, eds. Proc. of the 1st Biennial Conf. on Innovative Data Systems Research. Asilomar: Online Proceedings, 2003. 245..256.

[4] Carney D, Cetintemel U, Cherniack M, Convey C, Lee S, Seidman G,Stonebraker M, Tatbul N, Zdonik S. "Monitoring streams—a newclass of data management applications". In: Lochovsky FH, Wang S,Papadias D, eds. Proc. of the 28th Int'l Conf. on Very Large DataBases. Hong Kong: Morgan Kaufmann Publishers, 2002..

[5] Madden S, Shah M, Hellerstein JM, Raman V. "Continuously adaptive continuous queries over streams." In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM, 2002. 49to60.