# An Efficient Approach for Storage of Big Data Streams in Distributed Stream Processing Systems

Sultan Alshamrani[1], Quadri Waseem[2], Abdullah Alharbi[3], Wael Alosaimi[4], Hamza Turabieh[5], Hashem Alyami[6]
Faculty of College of Computers and Information Technology, Taif University, Kingdom of Saudi Arabia[1,3,4,5,6]
Independent Researcher Srinagar, Jammu and Kashmir, India[2]

***Abstract*—Besides, centralized managing, processing and querying, the storage is one of the important components of a big data management. There is always a huge requirement of storing immense volumes of heterogeneous data in different formats. In big data steam processing applications, the storage is given a priority and always plays a big role in historical data analysis. During stream processing, some of the incoming data and the intermediate results are always a good source of future samples. These samples can be used for the future evaluation to eliminate the numerous mistakes of storing and maintaining the big data streams. Hence, a big data stream application requires an efficient support for storage of historical queries. The researchers, scientist and academicians are working hard to develop a sophisticated mechanism that is needed for storage to keep the most useful data for the future references by means of stream archive storage. However, a stream processing system can't store the whole incoming stream data for future references. A technique is needed to get rid of the expired data and free the space for more incoming data in an archive storage. Hence keeping in view, the storage space limitation, integration issues and its associated cost, we try to optimize the stream archive storage and free more space for future data. The proposed enhanced algorithm will help to delete the obsolete data (retention or expired) and free the space for the new incoming data in a distributed platform. Our paper presents an Enhanced Time Expired Algorithm (ETEA) for stream archived storage in a distributed environment for removing the obsolete data based on time expiration and providing a space for the new incoming data for historical data analysis during the skew time (Hot Spots).We also evaluated the efficiency of our algorithm using the skew factor. The experimental results show that our approach is 98% efficient and fast than other conventional techniques.**

***Keywords*—Distributed stream databases; storage optimization; stream archive storage; time expiration**

## I. Introduction

Big data management is a way of centralized storing, managing, processing and querying the huge volume of different available data in numerous formats [1-5]. The traditional database technologies had failed to manage and control the flow of data which is overloaded with huge volume, variety, velocity and variability. However, alternative database technologies have played a vital role in solving "big data" issues of managing and processing. Their contribution has played a vital role in overall big data management [6]. As we know, big data computing has a huge demand for storage and processing [7]. Two types of processing can be done on big data. The first one is Batch processing and the Second one is Non-Batch processing of big data (real-time processing) [1] [8] [9], and the Non-Batch processing includes the real-time OLTP online transaction processing database management DBMS systems. They possess variable workloads, spike in traffic and are always dependent on shared nothing architecture besides using the main memory for the processing and scalability. Hence, they are best to maintain the acid guarantee of the transactions [9-16]. Stream processing engines (SPEs) are the engines that can generate huge and big data streams continuously on the fly in a cluster of commodity servers. Stream computing involves the computations for the analytic purpose. Stream processing engines (SPEs) [17] have achieved broad adoption in research and industry [18-19] and mainly focus on scalable cloud computing. In distributed stream processing systems, most of the stream-based applications are distributed naturally [20]. One of the difficult problems that need to address in distributed stream processing systems is a storage. The large size and a variety of data always creates a hurdle for an efficient big data storage [21-23]. Data Storage is one of the crucial processes of big data analytics for real-world applications. These real-world applications include scientific experiments, social networks, healthcare and e-business. Till now, only Amazon, Google, Apache and some companies had provided the big data storage solutions. However, the available big data storage technologies are not enough efficient in a sense to provide consistent, scalable, and available solutions for the continuously growing heterogeneous data [24]. Distributed Stream Processing Systems (DSPS's) has smartly evolved to store discovered patterns, analyzed data, and extracted knowledge from different data processing stages. The Stored data must be useful data, which must be well controlled, organized and indexed along with metadata or external knowledge. The main purpose of storing the data is to get historical data for future verification and tuning purposes [25-26]. The stored data is often used for later reference. There is always a capacity limitation associated with every archive storage system. None of the systems can store more data than its capacity. The capacity of each system is directly associated with cost. Therefore, a sophisticated mechanism is needed to find the obsolete data (retention or expired), delete that data and free space for more incoming data in archive storage. Our proposed algorithm will provide a solution for the space limitation for the stream archive storage by detecting and deleting the retention data and free the space for the new incoming stream data without adding more storage externally .The algorithm will be beneficial in a way to save the cost of extra storage and its associated issues of integrating. Our proposed algorithm will use the skew factor for the retention policy which will guarantee the strength of the approach.

Hence, we try to optimize the stream archive storage and evaluated the accessibility of our approach by implementing it with the YCSB benchmark. For our experiments, we have used synthetic data as stream data and has implemented it with H-store. In our implementation, we have modified the Time Expired Packets Algorithm (TE) [27] for the optimization of the stream archive storage in a distributed setup. In summary, this paper makes the following contributions as 1) We discuss some of the open issues related to storage of big data streams in distributed stream processing systems and also elaborates storage optimization for archived data in a distributed streams databases (DSDBM's) (Mentioned in Background Section). 2) We presented an Enhanced Time Expired Algorithm (ETEA), for stream archived storage in a distributed set of processing nodes. The algorithm will remove the obsolete data and will provide a space for the new incoming data for historical data analysis. 3) We also maintain the efficiency using Skew factor) in our experiments.

The rest of this paper is organized as follows: In Section 2, is a background, Section 3 is the literature review, Section 4 is related works, Section 5 is the introduction of our proposed algorithm, Section 6 is our detailed algorithm (Enhanced Time Expired Algorithm (ETEA), followed by Section 7 will depict our evaluation and Section 8 which gives our conclusion and future works.

## II. Background

Data Storage for stream processing is an important aspect for future enhancement. In big data steam processing applications, the incoming data and intermediate results may need to be stored to enable future analysis [28]. These applications require genuine support for storage and for historical queries. These required efforts help them for future analysis of historical data [29]. A lot of work has been done to optimize the stream archive storage. We have classified the stream archive storage optimization into three main subcategories which include 1) keeping most useful data. 2) Integrate Stored and Streaming Data (join live data). 3) Performance of storage manager.

### A. Need of Historic Data

Storage of Big Data Streams in Distributed Stream Processing Systems plays an important role in today's online stream world. One of its important pillars includes stream archive storage. As we know there is always a need for the storage of the intermediate data of stream processing in distributed setups for the enhancements, verification, future references and for tuning purposes [30]. Hence there is always a needed to keep the most useful data efficiently [28] for future analytics.

We try to highlight some of the open issues related to archive storage optimization.

### B. Open Issues of Storage Optimization

*1) Keeping most useful data:* For distributed stream databases, to keep the most useful data is always a key to storage optimization. An efficient storage algorithm is needed to keep the useful data and delete the obsolete data (retention or expired) from the archived storage. The optimizing protocol should be simple and automatic in nature. There is a huge need to keep the most useful data in order to get future analysis from the least stored streams [27-28] for enhancements.

*2) Integrate stored and streaming Data (join live data):* Careful management of live and historic data is a basic requirement for archive storage optimization. The routine task for most of the stream processing applications (on-line data mining) is a comparison of live data with historical data. These applications need seamless switching of both past and present data for the purpose of comparison within the same application. Hence, a uniform language is needed to deal with either type of data for seamless integration. Moreover, there is a demand [30-33] for automatically switching from historical to live data without manual intervention.

*3) Performance of storage manager:* The stream databases should have the capability to store, access and modify the state information efficiently and effectively. One of the primary concerns for any stream database is its storage and management. The protocols related to storage managers are regarded as an indictment of determining and preserving the storage for the future. Most of the research works pertaining to stream storage have been on the issues related to the storage manager .On the other hand, storage optimization and related techniques for live stream databases are a comparatively new arena of the storage and that too when its distributed. Thus, maintaining storage becomes a significant mission. Although storage has continued to be studied for decades, its maintenance is still at the infant stage of research. We have studied and mentioned some works related to archive storage optimization and some related work with references for further understanding to shape and enhance the basic protocols of a storage manager, its optimization and related algorithms [8][34].

The capacity limitation of the archive storage system is directly associated with the extra cost and integration issues. Those solutions which are dependent on cost are never considered as an idle solution. Therefore, we provide an alternate solution to storage capacity issue. The feasible solution is to delete the obsolete data (retention or expired) of archives and make a room for the new data for further future analysis. Conventional methods like First in First Out (FIFO) and other related solutions are less efficient to provide the best solution when the stream databases are under skew time (Hot tuples-when most of the users and servers are on max utilization).

## III. Literature Review

There is substantial literature available for the data storage for stream processing related to big data applications. Some of the research work done in the field of data storage management aims to improve storage, integrate the live data with archive data and provide correctness guarantee for big data stream processing.

In [8], Fred Douglis et al. proposed a storage system that optimizes not only reading and writing but the creation and deletion as well. Efficiency is achieved, by automating deletion based on relative retention values rather than requiring data to be deleted explicitly by an application. It works mostly on retention value functions, which effectively assign each data object a value that changes over time. In paper [28], Kirsten Hildrum et al. proposed an effective scheme for optimizing the placement of data within a distributed storage subsystem employing retention value functions. The goal is to keep the data of the highest overall value, while simultaneously balancing the read load to the file system. In paper [29], Nesime Tatbul, et al. proposed an S-Store which is designed to address the correctness aspect of a streaming application. In the paper, they prove the only way to achieve good performance is by tightly integrating storage management with the streaming infrastructure which supports correctness without serious performance degradation. The paper represents the exactly one processing, exactly one delivery, and transactional workflows. In paper [34], Irina Botan et al. proposed an optimized general-purpose storage management interface based on the parameters from the application requirement at different granularities. Using the interface and SMS (Storage Manager for Streams) can generate a customized storage manager for streaming applications. It uses information about the access patterns of streaming applications to tune and customize the performance of the storage manager. It efficiently handles time-critical tasks such as managing internal states of continuous query operators, traffic on the queues between operators, as well as providing storage support for shared computation and archived data. In Paper [8], Fred Douglis et al. proposed a storage system that optimizes not only the reading and writing but the creation and deletion as well. The papers use the concept of the relative value to retain the data items rather than deleting the data explicitly.

Therefore, keeping in view the necessity and the importance of the archive storage in distributed setups. It seems that there is a need for more research and a lot of work needs to be done for the archive storage and its optimization. We have designed an algorithm for the distributed stream processing engines which will help to optimize the stream archive storage by removing the expired data and free more space for incoming streams for the purpose of historical analysis during the skew time (hot tuples).

## IV. Related Works

For research and a commercial purpose, there are many big data storage and analysis models available in a market. The challenges of big data storage are widely understood. At present, a lot of work is going on related to datacenter storage. Storage and processing through different data centers are growing as fast as the big data itself. A huge amount of data is created by a variety of users and devices. For storing and processing big data, the data center is always needed to establish network infrastructure which helps to gather this rapidly generated data [23]. In another work [35], the authors highlighted and classify the components of the network that must be established for better storage and communication in data centers such as the original data network, the bridges and a datacenter. Another related study [36], identifies the issues in using big data through specific locations.

We have categorized some of the related works especially related to resource management and storage optimization for easy understanding.

### *A. Resource Management*

A lot of work has been done for storage load management of data stream processing systems. The proposed system uses the up-date queues for minimizing memory consumption and the impact of overload on QoS [37]. Another related research for resource allocation in the stream database includes the minimum spanning tree-based algorithms to discover and allocate the resources to meet real-time constraints [38].One more connected work aims to maximize the quality of the results in stream processing systems using overload management concept. This concept of overload management related to distribution in stream processing system uses the resource allocation technique. That includes the distributed algorithms for reallocating the system resources (i.e., CPU) based on their utilization [39]. Another related contribution [40] handles the query optimization, scheduling, resource allocation and especially the source availability. The proposed system interacts between resource availability and the approximation.

### *B. Storage Optimization*

Similarly, researchers investigated the storage optimization for data streams. One of the works includes a sluggish ladder queue that handles the long-running queries/continuous queries in real-time over a high-volume of data streams. Hence focuses on the latest data and control infinite streams overflow [41]. Some of the related work includes model integration of distributed stream processing system with state management. The model transitionally extends with sub-graphs, integrity constraints and consistency guarantees [42]. Some of the similar contributions include a high-performance stream processing engine I/O architecture which allowed the simultaneous persistence and communication of live and past (retrieved from storage) data streams [32]. Another related work [43] includes a system which constructs models and algorithms for overload prediction for heterogeneous data. The system scales up the performance and reduces the data loss without allocating additional servers.

Moreover, there has been a lot of works that deal with storage and is related to relational databases, data marts, data warehouses, and longer-term storage using Extract, Transform, Load (ETL) or Extract, Load, Transform (ELT) tools [15][21]. Another related work includes the development of an elastic cloud data storage system to support both OLTP and OLAP workloads efficiently within the same storage and processing system [44]. Enterprise Data Warehouse (EDW) traditional environment and its association with the data storage is another related work that needs a genuine consideration to enhance the big data storage [11][45]. Lastly, a review paper [46] has elaborated many data streams processing systems in depth that include their data models, continuous query processing, languages and query optimization.

## V. Proposed Algorithm

This section describes the Enhanced Time Expired Algorithm based on a skew time (Hot Spots) by utilizing the skew factor, generation time and expired time of the stream data.

## A. Problem Defnition

The problem of optimizing the archive storage in distributed stream databases is one of the big issues to tackle. A big concern in-stream archive storages is its limited storage space. One way to get rid of this problem is to increase the archive memory but that too has its limitations of cost and integration issues. Another way is to delete the expired data and make space for the new incoming streams. If there will be more space, more incoming data can be stored for future verification and tuning purposes. The storage space limitation has a direct impact on the storage and effects indirectly to historic data storage. Lots of techniques have been used to do the same work. Even a normal deletion technique would be an idle solution to free up the previous retention data and make space for the new incoming data based on normal retention policies as mentioned in [8]. However, these conventional solutions are not effective when it comes to the peak time (when most of the users are active and huge data is coming in). To our knowledge, none of the techniques had worked under the skew time. So a mechanism is needed which should automatically detect and delete the expired data and free the archive space for more incoming data for future historical data analysis under the skew time (Hot tuples-when most of the users and servers are on max utilization).

The problem of stream archive storage during peak hours needs a genuine optimized solution to automatically manage the storage space for huge incoming data by deleting the retention data and freeing up the space for huge incoming streams for historic analysis.

Thus, we define our algorithm to solve the problem of detecting and deleting the expired data based on the prescribed retention policy (Rp) which we keep 4 in our test environment (quarterly-4). A skew factor as Sf = {states*}, which includes the peak time of skews. Its value can be +ve or -ve based on the server and user load. The Tg* denotes a Time Generation of the stream and Et* denotes the Expire Time based on the retention policy. When retention value is found, the data is deleted and if not, the data is retained, and no action is taken till loop checks and continues to the whole archived storage.

## B. Enhanced Time Expired Algorithm (ETEA)

The Enhanced algorithm is derived from [27], a skew factor in addition to the generation time and the expired time of the streams is added. The skew factor represents the peak mode and the expired time represents the maximum allowable time for the stream to be available in an archive (based on the retention policy). The algorithm helps to recognize and remove the obsolete data (retention or expired) so as to free the space for more incoming data for archive storage in the future for historical data analysis. Therefore, it optimizes the archive storage. Based on positive and negative value criteria of time retention, the algorithm will discard expired data which is of no benefit and has occupied the archive space. Thus, freeing the archive's storage for more new incoming data.

The overview of an Enhanced Time Expired Algorithm (ETEA) is represented as:

**Algorithm: Enhanced Time Expired Algorithm (ETEA)**

```
1: is skew ← true // check for skew factor Sf
2: Get the “time generation” and generate “expire time” (Tg and Et)
3.Calculate the remaining Time Rt //based on retention policy
4: while skew do
5: Remove the retention stream //delete the stream based on retention policy
6: Update the value of skew
7: if not skew then
8: return false//no deletion done
9: end if
10: end while
11: return true//the system is still checking
```

## C. Our Contribution

The main contribution of this paper is that we propose an enhanced algorithm, called Enhanced Time Expired Algorithm (ETEA), to detect and delete the expire data automatically in a stream archive storage for distributed setup. Our algorithm is efficient and effective in detecting and deleting the retention data and free the space for new storage for future verification and tuning purposes.

During a stream processing in a distributed environment (distributed set of processing nodes) commonly known as Distributed Stream Processing Systems (DSPS's) for example OLTP, when a skew factor (Sf) is found, which represents the peak time for incoming data from different types of online servers and users. The proposed algorithm gets the Time Generation and Expire Time (Tg and Et) for each stream using a function: Get Time Generation (Tg) and Get Expire Time (Et). The Tg= initial time of the stream and Et=Expired time or retention policy time (it might be monthly or quarterly)-in our case we take it as quarterly for a test environment. When Rt Expired is found, the algorithm destroys the data stream in the archive using the following function: DELETE=Stream. If no Rt Expired match is found, no action is taken, then a second stream in the loop (Lp) is checked for the Rt Expired values onwards.

Our main contribution is our proposed algorithm for storage optimization (memory and cost efficient) and works efficiently during the skew time to find (detect) the expired data and delete the expired data based on the retention policy to release the memory for the new incoming stream data.

## VI. Algorithm Details

In this section, we explain the working of our proposed algorithm. As we know that many OLTP workloads are heavily skewed to “hot” tuples or ranges of the tuple. So, first, we try to identify the skew mode using the lightweight threshold mechanism based on several streams (transactions or stored procedures). For our experiments, we have assumed that if the maximum no of streams is 1000 per minute, we consider this as the maximum threshold. Once this threshold is achieved, a skew mode is triggered with a value representation. These values can either be 1 or 0 based on a condition whether the skew mode is true or false (+ve or -ve). If the value is found true, a skew factor is assigned with a timesheet which means more space is needed for the incoming data flow of streams. Sf = { states*}=1 or 0.The initial values for Time Generation decide the Expire Time for the streams based on the retention policy of four (4) months in our experiments, which means 4*30=120+2=(122) days. The data is deleted automatically once this value is found on matching. The whole process is repeated and cross-checked for any missing values until it checks the whole archive storage.

The Algorithm working is explained through below-mentioned phases:

- Phase 1: The working of the algorithm is based on the concept of checking for the skew factor (+ve/1 or -ve/0).
- Phase 2: Assigning the Time Generation and Expire Time for the streams. When the expired data is found, based on the retention policy, the long-stayed data is deleted automatically.
- Phase 3: The data is checked again for all streams if no match values of Expiration Time is found, the checking proceeds until it checks the whole archive storage.

The logic for (Sf) phase1: Sf = {states*}, which includes the peak time of skews. Its value can be +ve or -ve based on the server and user load. When the skew factor (Sf) is +ve, which represents the peak time during the online transactions of the distributed environment, the initial values are generated and set as 1 for +ve value and vice versa for 0 for -ve value.

The logic for (Tg and Et) phase2: The Tg* gets the Time Generation of the streams i.e. their initial time. Therfore based on it, Et* gets the Expire Time from the retention value (which is based on its policy). We are using Expire time (4 months in case of our experiments). Hence in our case it is calculated as 4*30=120+2=122 days. When this value (122) is found, the data is deleted automatically and if not, the data is retained, and no action is taken. Therefore, data of more than 122 days is deleted automatically.

The logic for (Lp) phase3: The data is checked again for all streams using a loop function, if no match values of Expiration Time is found, no action is taken, and the checking proceeds till it checks the whole archive storage is checked.

## VII. Performance Evaluation

In this section, we provide an experimental evaluation of the stream archive storage techniques using our proposed algorithm. The main goals of our experimental study are as follows:

- to show working of our Enhanced Time Expired Algorithm (ETEA) for stream archived storage in a distributed environment. The algorithm uses a skew factor based on hot spots and is compared against the state-of-the-art retention policy like conventional approaches First in First Out (FIFO), in terms of storage optimization (memory and cost efficient) for stream archive storage.
- to evaluate how effectively we can manage the memory space during the skew time.

We first investigate the impact of skews of incoming data rates. We found that there is a need for more archive storage for the stream data during peak time. Second, we consider the need for data deletion for skew. If more data will be deleted automatically, more space will be free for the future tuning purpose of historic data in archive stream storage. Thirdly, we detect the data automatically using our retention policy in the proposed algorithm. Fourthly, the expired data was deleted, and more space was available free. We can see from Fig. 1 and Fig. 2 that our algorithm had increased the effectiveness as well as memory optimization for stream archive storage. We observe all differences in performance (in terms of skew time and normal incoming data). Nevertheless, the conclusion is that we start with skew formulation, observe the impact of skews, provide the retention policy, detect the expired data to be deleted and lastly delete the expired data. Finally, we compare our proposed algorithm with one of the conventional approaches First in First Out (FIFO).

In Table I, we can see Time results for data deletion, which keeps increasing with the passage of time and Table II represents the values for Space (memory) results for data deletion).

Within a time of 3ms, the deletion was done with a value of 1 tuple deleted with an efficiency of 30.3 percent. The efficiency keeps increasing and decreasing with the time slots until it verifies the whole deletes of memory chunk (memory block).

In Table II, with the first deletion of 1 tuple, a space of 0.3 bits was released i.e. space(memory). The value of efficiency goes on increasing depending upon the memory released. The rest values of the table represent the respective space released with the efficiency.

In Fig. 1, we can see that in less time, the efficiency has increased, while as in Fig. 2, during the same time, more expired data is deleted, hence releasing the memory(space).In both cases (Fig. 1 and Fig. 2), which are related to efficiency improvement and memory improvement, the frequency is common. The Node values represent the nodes numbers, the time is represented in milliseconds (ms) and the deletion is measured in percentages in aggregate.

TABLE I. Block wise Time Results for Data Deletion

| | | Time | Deletion | Efficiency |
|---|---|---|---|---|
| Node Value | 1.00 | 3ms | 1.00 | 30.3 |
| | 2.00 | 6ms | 2.00 | 60.3 |
| | 4.00 | 1ms | 0.3 | 10.1 |
| | 6.00 | 9ms | 3.00 | 98.0 |

TABLE II. Block wise Space (memory) Results for Data Deletion

| | | Deletion | Space (Memory) | Efficiency |
|---|---|---|---|---|
| Node Value | 1.00 | 1.00 | 0.3 | 30.3 |
| | 2.00 | 2.00 | 0.6 | 60.3 |
| | 4.00 | 0.3 | 0.1 | 10.1 |
| | 6.00 | 3.00 | 6.0 | 98.0 |

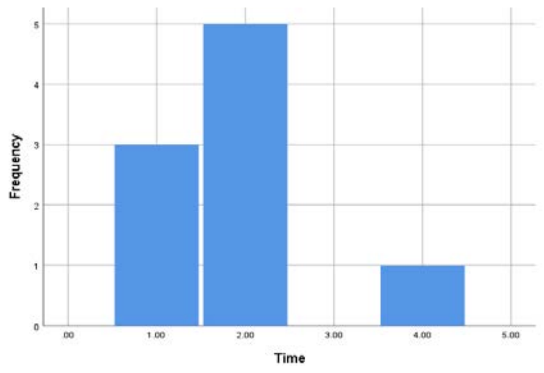

Fig. 1. Efficiency vs Time.

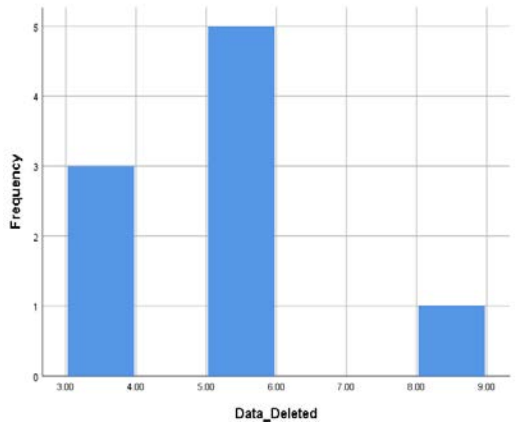

Fig. 2. Efficiency vs Memory (Data Deleted).

Next, we will first describe our experimental setup, then we will present our results.

## A. Experimental Setup

We implemented our ETEA algorithm as part of the H-store [47], mostly used for the OLTP workload. We monitor the flow of skews using its storage manager component and the memory usage was taken into consideration. We configured the monitoring component of H-store processing system to check the skew mode (hot tuple) and monitor the flow of incoming data into the archive storage.

In all our experiments, we have used a double-node set up for running on a window on an Intel Quad-Core Intel Xeon 3360 2.8GHz processor and 8GB of memory. To understand the impact of skew on an OLTP DBMS, we conducted a basic benchmark using the YCSB workload [48] on four (4) node H-Store clusters.

For this setup, we used a database with 2 million tuples (Each 1KB in size (~2GB in total) that are deployed on two (2) partitions.

*1) Workload:* In all experiments, we have used synthetically generated tuples (Streams). The input rates were set according to the desired level of threshold to be exerted on the system. The input is ordered by a time-based factor that decides the initial time of the streams and accordingly using the initial time. The expired time is calculated based on the retention value. The number of values can be altered and can range from 122 to any value depending on the experiment. The actual initial value of the streams does not have any significance, it is only what we measure in the experiments. To be able to control the processing cost of this query, we use the retention value to free the space which directly affects the cost of maintaining the more space unnecessarily for the expired data.

*2) Performance Metrics:* We have primarily used two performance metrics in our experiments:

- Effectiveness during Skews: Average effectiveness is computed on streams based on the no of tuples deleted and then, an overall average is computed across all the skews, which is measured in seconds.
- Storage (Memory optimization): Maximum memory deleted for the stream archive is recorded between output deliveries. Then we compute a maximum overall memory calculation based on deletion across a given run. Memory deleted is measured in the number of deleted tuples.

## B. Results

In Fig. 1 and in Fig. 2 we can see that the effectiveness, as well as the memory optimization for the stream archive storage during the skew time. The effectiveness is increased due to a continuous deletion of the expired data. The situation looks improved for all possible scenarios. As expected, during the skew time, the more expired data is removed, the more archive storage memory is released. The reason for this is the response to streams (keep it or delete it) method of our proposed algorithm, which avoids the need for redundancy in rechecking the memory chunk again. This result clearly shows that deleting expired data increase the archive storage for the new incoming streams. Our approach is 98% efficient and fast than other conventional techniques like First in First Out FIFO (which cannot be used for skew workloads). We have benchmarked our algorithm against FIFO in a skew environment. We had concluded with the results which prove our proposed algorithm is well suited for the skews with 98% efficiency and memory optimization. The details of the benchmark are mentioned.

Table III represents the detailed information about the ETEA. We observe the efficiency of ETEA (which works under the skews). We found that in less time, more expired data is deleted, and 98% efficiency is achieved which was not a case with the FIFO showing in Table IV. Fig. 3 and Fig. 4 presents the graphical representation of the efficiencies between the two algorithms.

TABLE III. EFFICIENCY UNDER SKEWS

| **Enhanced Time Expired Algorithm (ETEA)** | | | | | |
|---|---|---|---|---|---|
| | | **Time** | **Deletion** | **Space (Memory)** | **Efficiency** |
| Node Value | 3.00 | 2 | 28.6 | 28.6 | 98.0 |
| | 5.00 | 6 | 71.4 | 71.4 | 96.0 |
| | 6.00 | 9 | 100.0 | 100.0 | 98.0 |

TABLE IV. EFFICIENCY WITHOUT SKEWS

| **First in First out (FIFO)** | | | | | |
|---|---|---|---|---|---|
| | | **Time** | **Deletion** | **Space (Memory)** | **Efficiency** |
| Node Value | 3.00 | 2 | 18.0 | 28.6 | 43.6 |
| | 5.00 | 6 | 71.4 | 71.4 | 54.0 |
| | 6.00 | 9 | 100.0 | 100.0 | 54.0 |

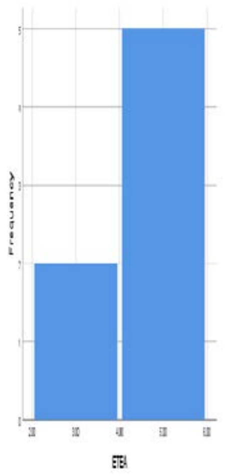

Fig. 3. Efficiency of ETEA.

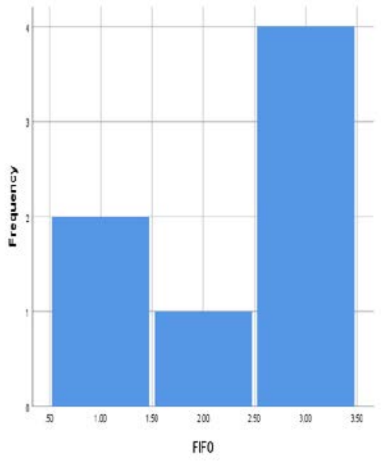

Fig. 4. Efficiency of FIFO.

## VIII. CONCLUSION AND FUTURE WORK

In this paper, we presented an algorithm, Enhanced Time Expired Algorithm (ETEA) for the distributed stream databases, which will help to optimize the stream archive storage and will detect and delete the expired data. Hence it provides the free space for new incoming streams for the purpose of historical data analysis during the skew time (hot tuples) in a distributed platform. Extensive evaluation on four (4) node cluster demonstrates the superiority of the approach compared to normal prior efforts which doesn't include the skew time (conventional retention techniques like First in First Out (FIFO). Hence, our solution is fully optimized based on effectiveness of memory and cost efficient factors of storage.

In the future, we will try to apply the other retention policies by using a machine learning algorithm (time series) to enhance the effectiveness and the performance of the stream archive storage. In our next series of optimizing the stream archive storage, we plan to design an architecture that will integrate the live data with historic data. Furthermore, we can also try the Pinwheel scheduling algorithm and Reduce-Max algorithm for the best results in our future experiments.

## REFERENCES

[1] Waseem, Q., Maarof, M. A., Idris, M. Y., & Nazir, A. A Taxonomy and Survey of Data Partitioning Algorithms for Big Data Distributed Systems. Micro-Electronics and Telecommunication Engineering, 447: (2020).

[2] Kaur, P., and Awal Adesh Monga. "Managing Big Data: A Step towards Huge Data Security." International Journal of Wireless and Microwave Technologies 2 (2016).

[3] Emani, C. K., Cullot, N., & Nicolle, C. (2015). Understandable big data: a sur-vey. Computer science review, 17, 70-81.

[4] Hu, Han, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. "Toward scalable systems for big data analytics: A technology tutorial." IEEE access 2 (2014): 652-687.

[5] M. Serafini, E. Mansour, A. Aboulnaga, K. Salem, T. Rafiq, and U. F. Minhas. Accordion: Elastic Scalability for Database Systems Supporting Distributed Transactions. Proc. VLDB Endow., 7(12):1035–1046, Aug. 2014.

[6] Moniruzzaman, A. B. M. "Newsql: Towards next-generation scalable rdbms for online transaction processing (oltp) for big data management." arXiv preprint arXiv:1411.7343 (2014).

[7] Kune, Raghavendra, et al. "The anatomy of big data computing." Software: Practice and Experience 46.1 (2016): 79-105.

[8] Douglis, Fred, et al. "Position: short object lifetimes require a delete-optimized storage system." Proceedings of the 11th workshop on ACM SIGOPS European workshop. 2004.

[9] Zhang, Hao, et al. "In-memory big data management and processing: A survey." IEEE Transactions on Knowledge and Data Engineering 27.7 (2015): 1920-1948.

[10] Bakshi, K., 2012. Considerations for big data: Architecture and approach conference. s.l., IEEE, pp. (1-7).

[11] Hartmann, T., Fouquet, F., Moawad, A., Rouvoy, R. and Le Traon, Y., 2019. GreyCat: Efficient what-if analytics for data in motion at scale. Information Systems journal.

[12] Hu, H., Wen, Y., Chua, T., & Li, X. (2014). Toward Scalable Systems for Big Data Analytics: A Technology Tutorial. IEEE Access, 2, 652-687.

[13] Singh, Dilpreet, and Chandan K. Reddy. "A survey on platforms for big data analytics." Journal of big data 2, no. 1 (2015).

[14] R. Taft. Elastic Database Systems. PhD thesis, MIT, Cambridge, 2017.

[15] Aubrey L. Tatarowicz, Carlo Curino, Evan P. C. Jones, and Sam Madden. Lookup tables: Fine-grained partitioning for distributed databases. In ICDE. IEEE Computer Society, 2012.

[16] Phansalkar, Shraddha, and Swati Ahirrao. "Survey of data partitioning algorithms for big data stores." In 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC), pp. 163-168. IEEE, 2016.

[17] Huang, Qun, and Patrick PC Lee. "Toward high-performance distributed stream processing via approximate fault tolerance." Proceedings of the VLDB Endowment 10.3 (2016): 73-84.

[18] Fu, Xinwei, et al. "Edgewise: a better stream processing engine for the edge." 2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19). 2019.

[19] Zhao, Xinwei, et al. "A taxonomy and survey of stream processing systems." Software Architecture for Big Data and the Cloud. Morgan Kaufmann, 2017. 183-206.

[20] Alshamrani, Sultan, et al. "High availability of data using Automatic Selection Algorithm (ASA) in distributed stream processing

systems." Bulletin of Electrical Engineering and Informatics 8.2 (2019): 690-698.

[21] Elgendy, N. and Elragal, A., 2014. Big data analytics: a literature review paper. s.l., Springer, cham, pp. 214-227.

[22] Zhong, R.Y., Newman, S.T., Huang, G.Q. and Lan, S., 2016. Big Data for supply chain management in the service and manufacturing sectors: Challenges, opportunities, and future perspectives. Computers & Industrial Engineering journal, pp. 572-591.

[23] Lv, Zhihan, et al. "Next-generation big data analytics: State of the art, challenges, and future research topics." IEEE Transactions on Industrial Informatics 13.4 (2017): 1891-1899.

[24] Siddiqa, Aisha, Ahmad Karim, and Abdullah Gani. "Big data storage technologies: a survey." Frontiers of Information Technology & Electronic Engineering 18.8 (2017): 1040-1070.

[25] Kamburugamuve, Supun, et al. "Survey of distributed stream processing for large stream sources." Grids Ucs Indiana Edu 2 (2013): 1-16.

[26] Isah, Haruna, et al. "A Survey of Distributed Data Stream Processing Frameworks." IEEE Access 7 (2019): 154300-154316.

[27] Abdullah, Mohammed B., and YazenS Sheet. "Refine Priority Queuing Scheduling Algorithm By Applying Time Expired Packets Algorithm." AL Rafdain Engineering Journal 20.2 (2012): 150-163.

[28] Kirsten Hildrum, Fred Douglis, Joel L. Wolf, Philip Yu, Lisa Fleischer, and Akshay Katta," Storage Optimization for Large-Scale Distributed Stream Processing Systems," ACM Transactions on Storage (TOS) TOS Homepage archive Volume 3 Issue 4, February 2008.

[29] Nesime Tatbul, Stan Zdonik, John Meehan, Cansu Aslantas, Michael Stonebraker, Kristin Tufte, Chris Giossi, Hong Quach,"Handling Shared, Mutable State in Stream Processing with Correctness Guarantees,"IEEE Data Engg.Bull ,2015.

[30] Antoniu, Gabriel, et al. "The Sigma Data Processing Architecture: Leveraging Future Data for Extreme-Scale Data Analytics to Enable High-Precision Decisions." (2018).

[31] Stonebraker, Michael, Uğur Çetintemel, and Stan Zdonik. "The 8 requirements of real-time stream processing." ACM Sigmod Record 34.4 (2005): 42-47.

[32] Sebepou, Zoe, and Kostas Magoutis. "Scalable storage support for data stream processing." 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). IEEE, 2010.

[33] Tatbul, Nesime, et al. "Handling Shared, Mutable State in Stream Processing with Correctness Guarantees." IEEE Data Eng. Bull. 38.4 (2015): 94-104.

[34] Irina Botan, Gustavo Alonso, Peter M. Fischer, Donald Kossmann, and Nesime Tatbu l,"Flexible and Scalable Storage Management for Data-intensive Stream Processing,"ACM Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology," Pages 934-945.

[35] Yi, X., Liu, F., Liu, J. and Jin, H., 2014. Building a network highway for big data: architecture and challenges. IEEE Network journal, Volume 28, pp. 5-13.

[36] H. Eszter, 2015.Is bigger always better? Potential biases of big data derived from social network sites. The ANNALS of the American Academy of Political and Social Science journal, Volume 659, pp. 63-76

[37] Moga, Alexandru. "UpStream: Storage-centric Load Management for Data Stream Processing Systems." Proceedings of the VLDB 2010 PhD Workshop. VLDB, 2010.

[38] L. Chen and G. Agrawal, "Resource Allocation in a Middleware for Streaming Data," in Proceedings of the 2Nd Workshop on Middle ware for Grid Computing. NY, USA: ACM, 2004.

[39] A. Tang, Z. Liu, C. Xia, and L. Zhang, Distributed Resource Allocation for Stream Data Processing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 91–100].

[40] Motwani, Rajeev, et al. "Query processing, resource management, and approximation ina data stream management system." CIDR 2003. Stanford InfoLab, 2002.

[41] Tang, Xiang-Hong, et al. "Storage Optimization for Query Processing over Data Streams." Journal of Chongqing University 9.2 (2010): 79-92.

[42] Affetti, Lorenzo, Alessandro Margara, and Gianpaolo Cugola. "FlowDB: Integrating stream processing and consistent state management." Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems. 2017.

[43] Nguyen, Nhan, et al. "Arion: A Model-Driven Middleware for Minimizing Data Loss in Stream Data Storage." 2017 IEEE 10th International Conference on Cloud Computing (CLOUD). IEEE, 2017.

[44] Cao, Yu, et al. "Es 2: A cloud data storage system for supporting both oltp and olap." 2011 IEEE 27th International Conference on Data Engineering. IEEE, 2011.

[45] Al-Shiakhli, Sarah. "Big Data Analytics: A Literature Review Perspective." (2019).

[46] Golab, Lukasz, and M. Tamer Özsu. "Issues in data stream management." ACM Sigmod Record 32.2 (2003): 5-14.

[47] Kallman, Robert, et al. "H-store: a high-performance, distributed main memory transaction processing system." Proceedings of the VLDB Endowment 1.2 (2008): 1496-1499.

[48] Cooper, Brian F., et al. "Benchmarking cloud serving systems with YCSB." Proceedings of the 1st ACM symposium on Cloud computing. 2010.