IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 9, 2021, accepted March 1, 2021, date of publication March 8, 2021, date of current version March 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3064202

An Efficient Data Access Approach With Queue and Stack in Optimized Hybrid Join

OMER AZIZ, TAYYABA ANEES, AND ERUM MEHMOOD
School of Systems and Technology, University of Management and Technology, Lahore 54770, Pakistan
Corresponding author: Tayyaba Anees (tayyaba.anees@umt.edu.pk)

ABSTRACT As rapid decision making in business organizations gain in popularity, the complexity and adaptability of extract, transform, and load (ETL) process of near real-time data warehousing has dramatically increased. The most important part of near real-time data warehouse is to feed new data from different data sources on near-real-time basis. However, this new data is not in the format of the data warehouse therefore, it needs to be transformed into the required format by using transformation algorithms which is essential part of ETL process. A semi-stream join algorithm is required to implement this transformation, for this purpose a HYBRIDJOIN (hybrid join) algorithm has been presented in the literature. However, major design issue with this algorithm is that it uses a single buffer to load the disk partitions and therefore, the algorithm has to wait until the next disk partition overwrites the exiting partition in the disk buffer. As the cost of loading disk partition into disk buffer is the major cost of overall algorithm processing cost, this leaves the performance of algorithm sub-optimal. Moreover, existing approaches only considering the oldest key join attributes for finding the matches with master data and maintaining the Queue of key join attribute. However, performance can be improved if recent and oldest attributes process in parallel. This article addresses the limitation of HYBRIDJOIN by presenting two optimized new algorithms named: Parallel-Hybrid Join (P-HYBRIDJOIN) and Hybrid Join with Queue and Stack (QaS-HYBRIDJOIN). Proposed algorithms aim to reduce major processing cost that is disk I/O as well as to increase number of matching stream tuples. Both of these algorithms perform significantly better in terms of throughput and number of matching tuples as compared to existing approaches. Performance analysis and cost model for proposed algorithms show the best performance using intermittent stream data under limited resources.

INDEX TERMS Near-real-time data warehouse, semi-stream join, optimized hybrid join, queue and stack.

I. INTRODUCTION

Business world has become the global village with many companies competing with one another to improve their resource management and business intelligence on basis of real-time data warehouse (RTDW) [1]–[3]. Decision support systems are dependent on RTDW, Enterprise Service Bus (ESB) [4] applications and big data [5]: a repository of complex and large data that can be analyzed for decision making. It is difficult for common business applications to process such data [6] on real-time basis. Main challenge in real-time big data is processing of data from different sources which can cause unexpected and unknown faults in information system if not handled properly [7]. These faults can be removed by using emerging techniques that can process the real-time big data efficiently and effectively. Business intelligence is dependent on latest data warehouse technology that is near real-time data warehouse [8]. The growing need of reporting and analysis emerge the idea of RTDW which appears as tool for better business forecasting [9]. Moreover, as size of historical data is increasing every year for each company [10]–[12], there is need to perform historical analysis to improve business progress of the companies [13], [14]. The general architecture of RTDW consists of three components.

The first is Data Sources hosting the data production systems that populate the data warehouse, second is an intermediate Data Processing Area (DPA) where the cleaning and transformation of the data takes place, and last is Data Warehouse(DW) to load the transformed data. General architecture of RTDW with three components is presented in Figure 1.

The associate editor coordinating the review of this manuscript and approving it for publication was Mehedi Masud.

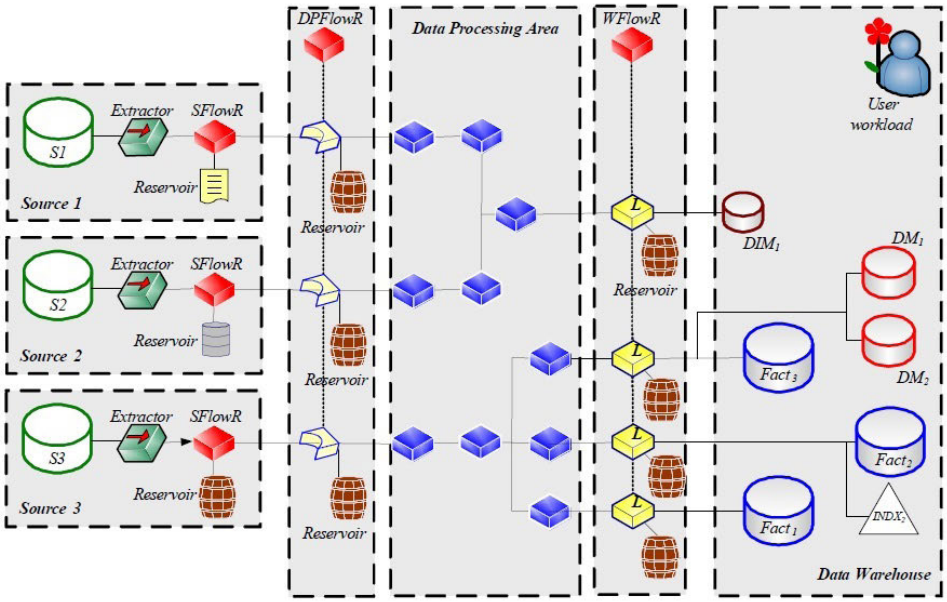

FIGURE 1. General Architecture for Near Real-Time Data Warehouse [15].

Many services need to organize data at a scale too large for a traditional database include: web analytics, social networks and smart e-commerce [16]. Complexity increases as scale and demand increase. A different approach is required [17] as simplicity and scalability not mutually exclusive. Big data systems utilize many devices executing in parallel to process and store data, which leads to basic challenges unfamiliar to most developers [18], [19]. Moreover, these days big data is considered as the only source for business intelligence [20]–[22].

Big data needs to be captured in form of data streams. The coherent continuous collection of attributes is called data stream which is intermittent in nature with variant velocity for near RTDW. As industry is moving towards RTDW for better decisions [6], [23]–[25] where records are extracted from intermittent stream and master data. These records are not in the format required by near RTDW, therefore, transformation phase is required to convert the input data into target format [26] and then loaded into the data warehouse [27]. For real-time stream transformation, all incoming data rows of stream need to be kept in memory which appears to be a big challenge because of fast speed and large volume of the continuous data [28]–[31]. The data stream arrival rate varies depending on the nature of business transaction on transaction outlets. The velocity is high in transaction rush hours while it is low in working hours of the buyer. Both sources of data have different arrival rates before transformation, therefore, risk of stream loss becomes another big challenge [32]. Algorithms designed to join stream with disk data are generally referred as semi-stream join algorithms. Figure 2 shows the basic model of semi-stream join process [33].

The data stream management system(DSMS) is the tool designated for proper data stream management [35].

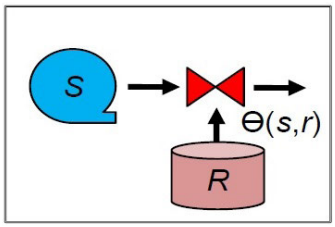

FIGURE 2. Graphical representation of semi-stream join [34].

Semi-stream data management is considered as an important area for research where researchers have contributed a lot for data transformation phase. This transformation is performed during ETL(Extract,Transform,Load) phase of data warehouse. The joining of semi-stream data remained under consideration of researchers as this area has evolved in last decade. This journey was started from processing hundreds of records per second with exponential raise to process millions of records per second [32], however, still there is need of further optimized solutions. Various fields require real-time semi-stream join operation: DW, risk management, online shopping, IoT, enterprise resource planning, supply chain management and, data retrieval and manipulation [36]–[54].

To achieve optimal performance of semi-stream join process, loading of disk partitions need to work in parallel to join operator as well as newest and oldest streams can be given equal opportunity for join operation. In this paper, we propose two optimized new algorithms called Parallel Hybrid Join (P-HYBRIDJOIN) and Hybrid Join with Queue and Stack (QaS-HYBRIDJOIN) by introducing two disk buffers (working sequentially/Parallel) and a component called Queue and Stack which results in finding matches for newest streams

along with oldest. These approaches minimizes the disk I/O cost and ultimately improves the service rate.

Later sections present: detailed discussion on problems and research gaps present in existing semi-stream join algorithms and, proposed methodology that can help to improve the efficiency of existing joins. The results and discussion presented in later sections justify how the proposed work is better than existing work. Future directions may help to make further improvements.

II. LITERATURE REVIEW

A number of algorithms have been proposed to process real-time stream in an efficient way [55]–[63] for RTDW [64]. During processing stream with disk data, the data is coming from two sources. From one source data is coming at quick arrival rate while the other is permanent disk data coming at slow arrival rate due to high I/O cost. Algorithms designed for in-memory join of stream with disk data are known as semi-stream algorithms [54], [65]–[78]. Working and limitation of most relevant semi-stream algorithms have been presented in this section.

A parallel semi-stream similarity join (S3J) algorithm proposed by [79], combines a disk-intensive queue-based semi-stream join approach with a CPU-intensive similarity matching algorithm to utilize disk and CPU optimally.This algorithm supports parallel execution to utilize modern multi-core CPUs and minimizes the memory footprint. Frequent join of online stream of updates and a disk-resident table of historical data are the basis for ETL transformations. In this context, another innovative Semi-Streaming Index Join (SSIJ) algorithm proposed by [80], that increases the efficiency of join operation by buffering online stream of updates and then accordingly selecting how to best amortize expensive disk sought for blocks of the stored relation among a large number of stream tuples. Furthermore, SSIJ appears to be able to adjust to variable attribute of the stream (i.e. arrival rate, data distribution) by dynamically tuning of memory between the cached relation blocks and the stream.

The Real-time Transport Protocol (RTP) and its related standards defined a re-transmission packet format [81] and a way to give feedback via Negative ACKnowledge (NACK) packets for data that had been lost. In one embodiment, a unicast RTP repair session was associated with a main Source Specific Multicast (SSM) multicast session. Real-time Transport Control Protocol (RTCP) NACK packets were then used for feedback to a SSM feedback target address. This dynamically instantiated unicast RTP repair for multicast sessions. The repair scheme could be used for repairing multicast channels or joining new multicast channels.

Another semi-stream join algorithm, Indexed Nested Loop Join (INLJ) [66], has the ability to work with intermittent data but with low throughput. MESHJOIN (Mesh join) algorithm [53], [54] appears to process stream data with high throughput. It divides the permanent disk data R into partition generating efficient results for continuous stream at high speed. This algorithm uses limited memory and is a candidate for a resource-aware system setup. However, performance of MESHJOIN always remain inversely proportional to the size of disk based master data set leaving this algorithm sub-optimal. Moreover, MESHJOIN is not very efficient in managing the intermittent stream.

An efficient algorithm R-MESHJOIN (R-Mesh Join) [65] has been introduced to overcome the inadequate memory distribution of MESHJOIN by introducing a new memory architecture. In this memory architecture the size of the permanent disk data would not affect the disk buffer size. It simplified the complex architecture of MESHJOIN by removing dependency between disk buffer and size of permanent data R. However, it is not efficient in dealing with variable speed arrival rate of stream data. Secondly the disk I/O cost to load permanent data is also high.

A new approach for stream processing has been introduced in another algorithm CACHEJOIN, which performed better than MESHJOIN if parts of non-stream disk based master data set are used with different frequencies. In order to quantify the performance differences, it compares both algorithms using a synthetic data set with a known skewed distribution [82].

Another stream join processing model has been introduced by [83] which is efficient for batch processing and also handling network delay issues. Two distributed join methods have been introduced in this model: (1) simple join, in which full tuples forwarded to the query processing site and (2) semijoin-based join, in which partial tuples have been forwarded.

Map Reduce stayed an important method that dealt with semi-structured or unstructured big data files, however, querying data mostly needed a Join procedure to accumulate the desired result from multiple huge files [84]. Indexing in other hand, remains the best way to ease the access to specific record(s) in a timely manner. Partition-based algorithm added its contribution to the intermittency in streams of join algorithm [85]. The algorithm has been focusing on the analysis of stream records backlog.

Another recently developed algorithm, block-nested loop join (BNLJ) [86], eliminates needles intermediate writes on disk saving the buffer sizes. However, this approach results in additional processing time.

Flexibility and self-adaptivity are important to real-time join processing in a parallel shared-nothing environment. Join-Matrix is a high-performance model on distributed stream joins and supports arbitrary join predicates [87]. Extensive experiments have been done on different kind of join workload and showed high competence comparing with baseline systems on benchmark. Matrix-based scheme (Join-Matrix) could perfectly support distributed stream joins, especially for arbitrary join predicates, because it guaranteed any tuples from two streams to meet with each other [88]. However,the dynamics and unpredictability features of stream required quick actions on scheme changing. Otherwise, they might lead to degradation of system

throughput and increment of processing latency with the waste of system resources, such as CPUs and Memories.

Famous semi-stream join algorithm HYBRIDJOIN and its versions [89]–[93] perform significantly efficient for intermittent stream data. Detailed working of all versions of HYBRIDJOIN is presented in the following section. However, join operation in these approaches need to wait for the time spend on loading the disk buffer partition in memory and newest stream also waits for its match as algorithms consider only oldest streams first. We overcome these gaps in this study by developing two modifications in existing versions of HYBRIDJOIN algorithms. This article develops and evaluates P-HYBRIDJOIN and QaS-HYBRIDJOIN algorithms improving throughput considerably.

III. EXISTING VERSIONS OF HYBRIDJOIN AND PROBLEM DEFINITION

HYBRIDJOIN [89] is a robust algorithm that has been particularly designed to deal with bursty stream data, combining the two approaches MESHJOIN and INLJ. unlike MESHJOIN, throughput of HYBRIDJOIN is not inversely proportional to disk size and algorithm uses indexes to access disk partitions, consequently perform better.

According to market survey the data coming in stream is 80/20 nature. The 20 percents of the products are generating 80 percents of the stream data. This survey brought an idea of further improving the performance of the algorithm by keeping the frequent data into the memory. X-HYBRIDJOIN(Extended Hybrid Join) [90] has been designed to support this idea, which shows better performance as compared to HYBRIDJOIN. The X-HYBRIDJOIN Join divided the disk buffer into two parts, one contained the most frequently matched records while the other part loads the disk data into it by discarding the previous data. There are two parts of the disk buffer: swappable and non-swappable. Non-swappable contains the matched records whereas swappable loads at each iteration of the algorithm probing phase.

Optimised X-HYBRIDJOIN [91] algorithm assumed that the disk data accessing is done on the frequency of data. Which is not the case at every iteration because when the data is accessed from unsorted index of the data, the performance of the algorithm decreases. This algorithm introduced two phases: stream probing phase and disk probing phase. Stream probing phase deals with frequently occurring stream unlike X-HYBRIDJOIN resulting in better performance.

A variant of X-HYBRIDJOIN is Tuned X-HYBRIDJOIN [92], assigns optimal division of memory to the components of X-HYBRIDJOIN. After proper tuning, this algorithm performs significantly better than X-HYBRIDJOIN.

Optimised HYBRIDJOIN [93] has further improved the performance with introduction of two buffers for loading from master data in parallel. It has improved performance because of less wait time by algorithm while one buffer is in probing phase the second buffer is loading phase from master data. Consequently, second buffer is in probing phase while first buffer goes for loading phase. This way the disk loading cost is decreased, however the algorithm works on an idea of oldest join key attribute, the key attribute remain longer in the memory have more matches in probing phase. Memory architecture for Optimised HYBRIDJOIN is shown in Figure 6.

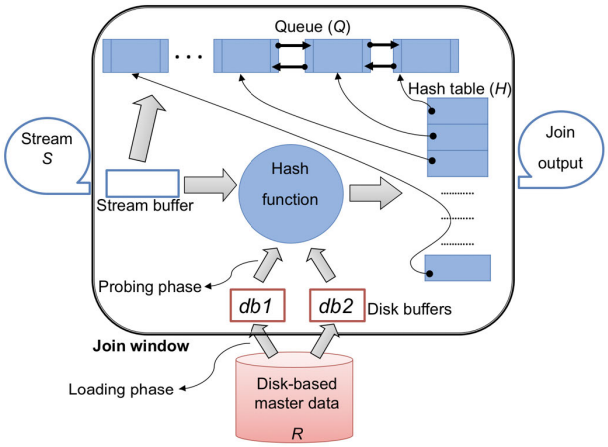

FIGURE 3. Memory Architecture for Optimized HYBRIDJOIN.

It has been identified that, the recent join attribute can have chances of being more matches in probing phase. This appeared to be most neglected concept in all previous semi-stream join algorithms which needs to be addressed. Our proposed methodology addresses identified gap by introducing a new component of Stack and Queue in a new algorithm QaS-HYBRIDJOIN. Proposed algorithm takes both fields as join key attribute the most recent and oldest key attribute. Our experimental results show that service rate is improved significantly in case of Qas-HYBRIDJOIN.

Comparison of existing versions of HYBRIDJOIN in terms of memory components for stream management and disk data loading, tuning and service rate is shown in Table 1. Service rates for one memory setting (i.e, combination of 4 million disk tuples with 50MB memory reserved for stream placement and join process) for relevant algorithms have been presented in Table 1.

Most of the versions of HYBRIDJOIN use only one disk buffer making partition loading process sub-optimal. Furthermore, nearly all algorithms have been evaluated under limited resources except Optimised X-HYBRIDJOIN. Comparative analysis presented in Table 1 clearly shows that existing contributions are novel and accurately perform better than previous studies. The next sections present the proposed methodology with the experimental results and discussion.

IV. PROPOSED METHODOLOGY

In this modification, proposed P-HYBRIDJOIN algorithm introduces the new disk buffer (db2), join window now having two disk buffers (db1 and db2) working in parallel for join operation. In addition to that, a new component QaS has been introduced making join operation further optimized with the development of QaS-HYBRIDJOIN.

TABLE 1. Comparison of all Versions of HYBRIDJOIN (Semi-stream join algorithm).

				Components									
				Stream Management				Disk Data Loading				Tuning	Service Rate
Article	Algorithm	Year	Approach	Stream Buffer	Queue	Stack	Hash Table	No. of Disk Buffers	Sequential to Join Operator	Parallel with Join Operator	Synchronous	Under Limited Resources	4 million disk tuples, 50MB memory (tuples/sec)
[89]	HYBRIDJOIN	2011	Loads only useful part of disk-based relation R into memory and deals with intermittent stream effectively	✓	✓	x	✓	1	✓	x	x	✓	$\approx 4x10^3$
[90]	X-HYBRIDJOIN (Extended Hybrid Join)	2011	Improves performance by introducing non-swappable part of the disk buffer	✓	✓	x	✓	1	✓	x	x	✓	$\approx 5x10^3$
[91]	Optimised X-HYBRIDJOIN	2012	Two phases of this algorithms work in parallel with the swappable and non-swapppable parts of disk buffer improving performance	✓	✓	x	✓	1	✓	x	x	x	$\approx 10^4$
[92]	Tuned X-HYBRIDJOIN	2013	Same number of components as of X-HYBRIDJOIN but different memory for both swappable and non-swappable parts of disk buffer	✓	✓	x	✓	1	✓	x	x	✓	$\approx 6x10^3$
[93]	Optimised HYBRID-JOIN	2019	Loads partitions of disk-based relation R into two disk buffers alternatively causing join operation without any wait, fully exploiting CPU and memory resources	✓	✓	x	✓	2	✓	x	x	✓	$\approx 6x10^3$
This Article (Contribution 1)	P-HYBRIDJOIN (Parallel-Hybrid Join)	2021	Join is performed parallel on oldest index from queue to fill the both buffers synchronously, significantly improving the performance	✓	✓	x	✓	2	x	✓	✓	✓	$\approx 7x10^6$
This Article (Contribution 2)	QaS-HYBRIDJOIN (Hybrid Join with Queue and Stack)	2021	Join is performed parallel on oldest index from queue using first disk buffer and newest index from stack using second disk buffer by introducing a new component QaS synchronously, significantly improving the performance	✓	✓	✓	✓	2	x	✓	✓	✓	$\approx 8x10^6$

Parallel working of probing and loading significantly improves the service rate. Figure 4 shows the context model of proposed algorithms. Memory architectures, work flows and algorithms of proposed approaches are presented in the following subsections.

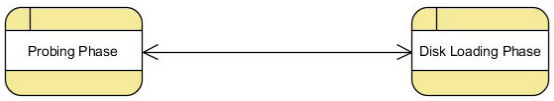

FIGURE 4. Context Diagram of QaS-HYBRIDJOIN.

A. P-HYBRIDJOIN ARCHITECTURE

Memory architecture for P-HYBRIDJOIN is presented in Figure 5. During the probing phase of db1, this algorithm loads db2 into the memory. After it finishes probing phase with db1, db2 becomes available to the join operator. During this cycle, the join operator performs probing with db2, meanwhile algorithm loads the next partition into db1. This way join operation doesn't need to wait for loading of next partition into memory.

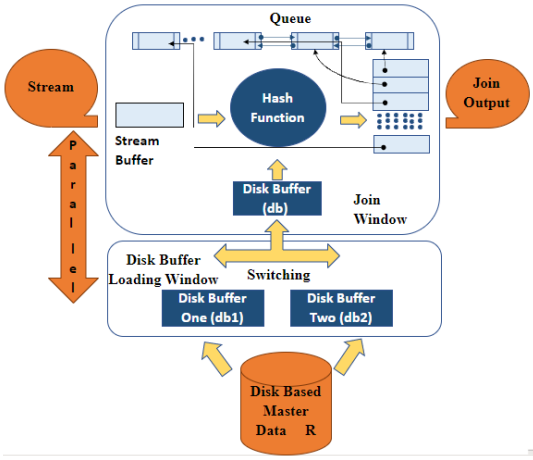

FIGURE 5. Memory Architecture for P-HYBRIDJOIN.

B. QaS-HYBRIDJOIN ARCHITECTURE

In order to pick oldest and recent stream tuples for join operation, a new component has been introduced in QaS-HYBRIDJOIN: queue and stack (QaS), replacing the queue (Q) component of the P-HYBRIDJOIN as shown

in Figure 6. QaS-HYBRIDJOIN shares the remaining component from P-HYBRIDJOIN. This new memory component QaS is implemented as double link list. After including QaS, db1 is set to use the oldest index attribute key from Q, while db2 uses the latest index attribute key from another double link list which behaves like stack named as S. If a match is found, join process is applied and the output is generated. It can be stated that the oldest attribute of QaS is used as an index to fill db1 and the most recent attribute of QaS will be used as an index to fill db2.

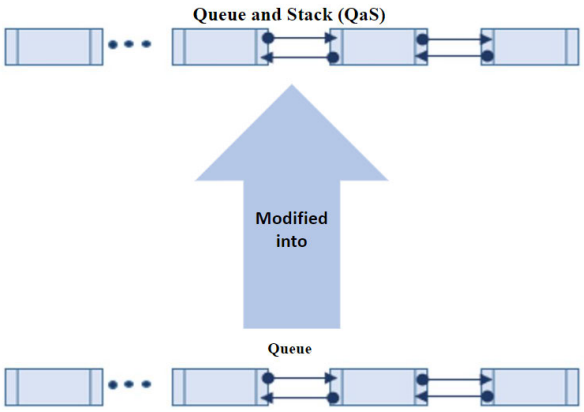

FIGURE 6. Modified Component for QaS-HYBRIDJOIN.

Three separate sub-programs have been executed in parallel, one is the stream generator that produces the stream of continuous records on zipfian's law of distribution (80/20 rule). The stream buffer S is filled from the stream generator that is used to fill QaS. QaS is working in hybrid nature: in the form of queue as well as stack. Hash table H also has been filled from the stream buffer S. In the mean time the join operation executes the join window, also probing the current disk buffer either db1 or db2 with hash table H.

Figure 7 shows the flow chart of disk buffer loading window in the presence of QaS of proposed algorithm. The QaS-HYBRIDJOIN algorithm contains two disk buffers db1 and db2 with the new memory component QaS as double link list in data structure. The db1 has used the oldest index attribute key from double link list as queue Q, while the db2 has used the latest index attribute key from double link list as stack S. db1 and db2 data is transformed and deleted with join operation in Join Window.

Figure 8 presents the flow chart of Join Window of QaS-HYBRIDJOIN. The current buffer is either db1 or db2 depending on recent disk data loading. The probing phase start matching and deletion of data loaded in disk loading phase and set the state of the buffer to busy.

The completion process of probing phase sets the buffer state to empty. The Empty buffer are available for data loading from disk partition R.

Data flow diagram of QaS-HYBRIDJOIN is shown in Figure 9. It is clearly depicted from Figure that db2 is in the process of data loading from master data Relation R while db1 is in join window. similarly, the db2 is in join window after the db1 finishes data join and start doing data loading. After adopting this new architecture, proposed QaS-HYBRIDJOIN algorithm does not put join operation on un-necessary wait unlike other Hybrid Join algorithms.

C. P-HYBRIDJOIN ALGORITHM

Two algorithms have been designed to fully implement proposed idea. Algorithm 1 shows the working of Join Window for P-HYBRIDJOIN.

Algorithm 1 P-HYBRIDJOIN/QaS-HYBRIDJOIN

```
1:  procedure JOIN WINDOW
2:      Input: A master data R with an index on join attribute
    and Stream of updates S
3:      Output: S ⋈ R
4:      Parameters: w records of S and a partition of R
5:      Method:
6:      h_s ← w
7:      while true do
8:          if stream available then
9:              Read: w records from S and load them in H,
    load keys of records in Q
10:             w ← 0
11:         end if
12:         Check: available disk buffer (db1) or (db2)
13:         Load: current disk buffer (db1) or (db2) for prob-
    ing phase of H
14:         Mark: current disk buffer (db1) or (db2) to busy
    state
15:         for each record r in chosen partition do
16:             if r ∈ H then
17:                 Output: r ⋈ H
18:                 Delete: matching records from H and Q or
    QaS
19:                 w = w + no of matching records in H
20:             end if
21:         end for
22:         Mark: current disk buffer (db1) or (db2) to empty
    state
23:     end while
24: end procedure
```

Line numbers 8-11 from Algorithm 1 show that if stream of Q is available, load w records from stream to the H and set w to zero. Line numbers 12-14 present that any one of the two disk buffers is available for join window, then the available buffer either db1 or db2 will be loaded to current buffer and buffer state will be set to busy. Lines 15-21 show that the algorithm iterates and matches one by one r from current buffer in H and if the r is found in H, it deletes r from H and Q, then adds the total matching records to w. In line 22 state of db1 or db2 is set to empty. While loop shows the unlimited execution of algorithm as streams are continuously generating.

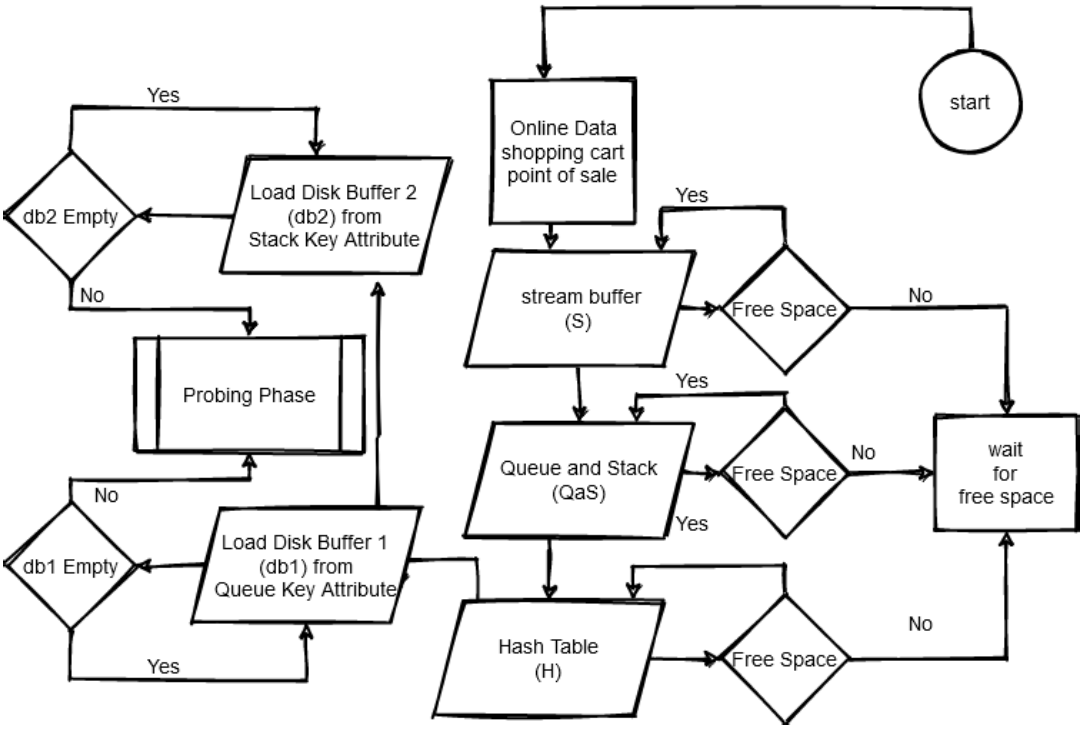

FIGURE 7. Flow Chart of Disk Buffer Loading Window with queue and stack (QaS).

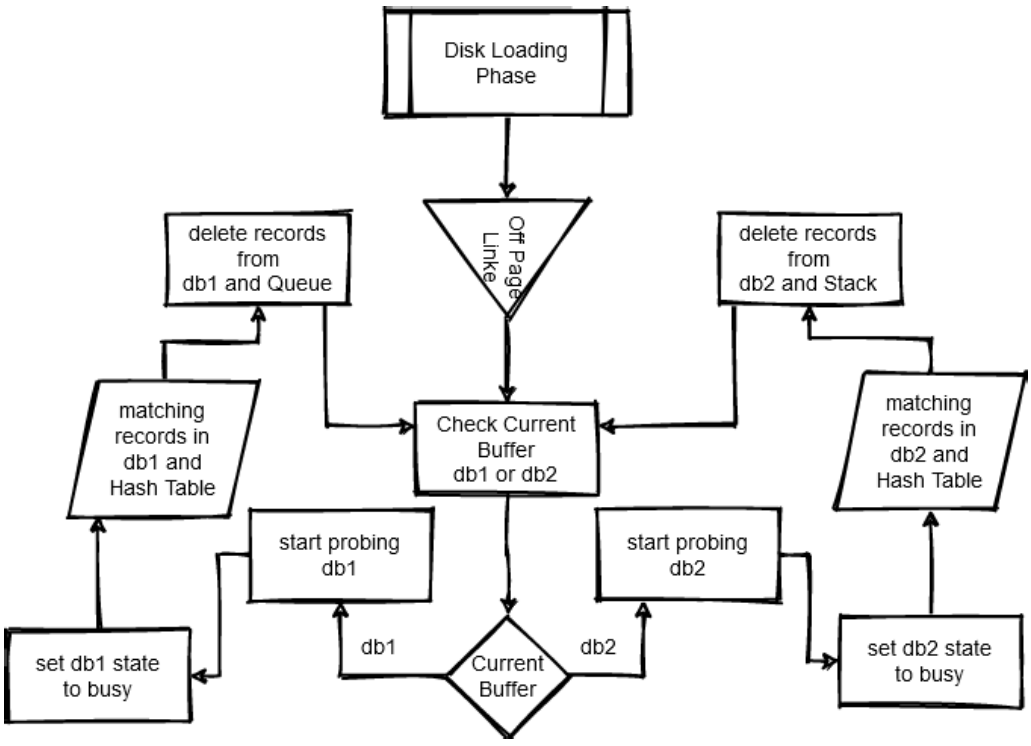

FIGURE 8. Flow Chart of Join Window with Queue and Stack(QaS).

Algorithm 2 presents the working of Disk Buffer Loading Window of P-HYBRIDJOIN. Lines 7-12 show that algorithm checks for empty disk buffer either db1 or db2. If the empty buffer is db1, it selects the oldest join key attribute from Q and loads partition p from master data R into db1 and sets the db1 state to full. Alternatively Lines 13-18 are used

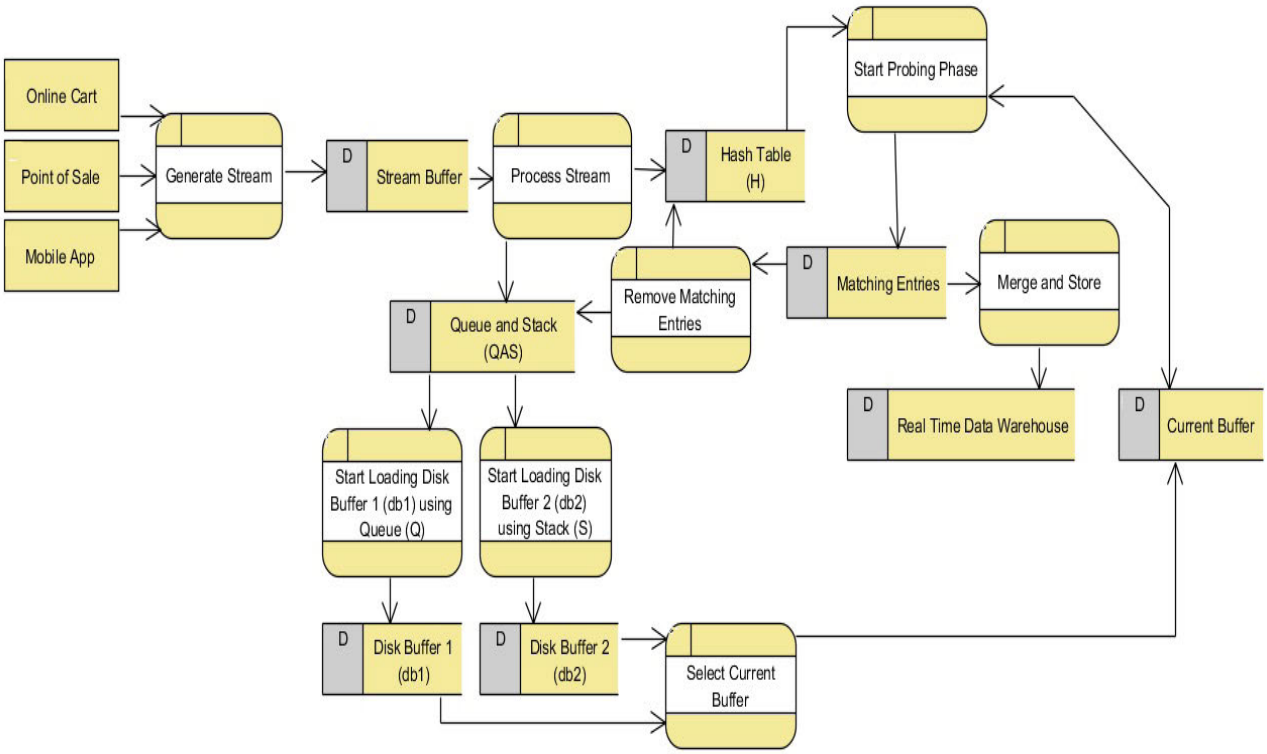

FIGURE 9. Data Flow Diagram QaS-HYBRIDJOIN.

Algorithm 2 P-HYBRIDJOIN

```
1:  procedure DISK BUFFER LOADING WINDOW
2:      Input: A master data R based on oldest index from Q
3:      Output: (db1) or (db2)
4:      Parameters: a partition of master data R
5:      Method:
6:      while true do
7:          if (db1) state is empty then
8:              Get: the oldest key attribute as index from Q
9:              Mark: (db1) state to busy
10:             Load: partition p from R in (db1) on oldest
    key attribute
11:             Mark: (db1) state to full
12:         end if
13:         if (db2) state is empty then
14:             Get: the oldest key attribute as index from Q
15:             Mark: (db2) state to busy
16:             Load: partition p from R in (db2) on oldest
    key attribute
17:             Mark: (db2) state to full
18:         end if
19:     end while
20: end procedure
```

to show that if the empty buffer is db2 then it selects the oldest join key attribute from Q and loads disk partition p from master data R. and sets the db2 state to full.

Algorithms 1 and 2 work in parallel for two cases, In the first case while db1 is in Disk Buffer Loading Window, meanwhile the db2 is in Join Window of P-HYBRIDJOIN and opposite for other case.

D. QaS-HYBRIDJOIN ALGORITHM

Although use of two disk buffers in P-HYBRIDJOIN improves disk I/O cost significantly, it is not applicable for recent and oldest semi-stream join operation simultaneously. To enable join operation for recent and oldest keys together we proposed another improvement called QaS-HYBRIDJOIN. Algorithm 1 shows the working of Join Window for QaS-HYBRIDJOIN where keys of records from S have been loaded into QaS unlike Q. Similarly line 18 shows the deletion of matching records from QaS.

Algorithm 3 presents the working of Disk Buffer Loading Window of QaS-HYBRIDJOIN. Lines 7-12 show that algorithm checks for empty disk buffer either db1 or db2. If empty buffer is db1, it selects the oldest join key attribute from QaS and loads partition p from master data R into db1 and sets the db1 state to full. Alternatively Lines 13-18 are used to show that if the empty buffer is db2 then it selects the recent join key attribute from QaS, loads disk partition p from master data R and sets the db2 state to full.

Both algorithms work in parallel for two cases, In the first case while db1 is in laoding phase of QaS-HYBRIDJOIN algorithm, meanwhile the db2 is in probing phase of QaS-HYBRIDJOIN and inversely for other case.

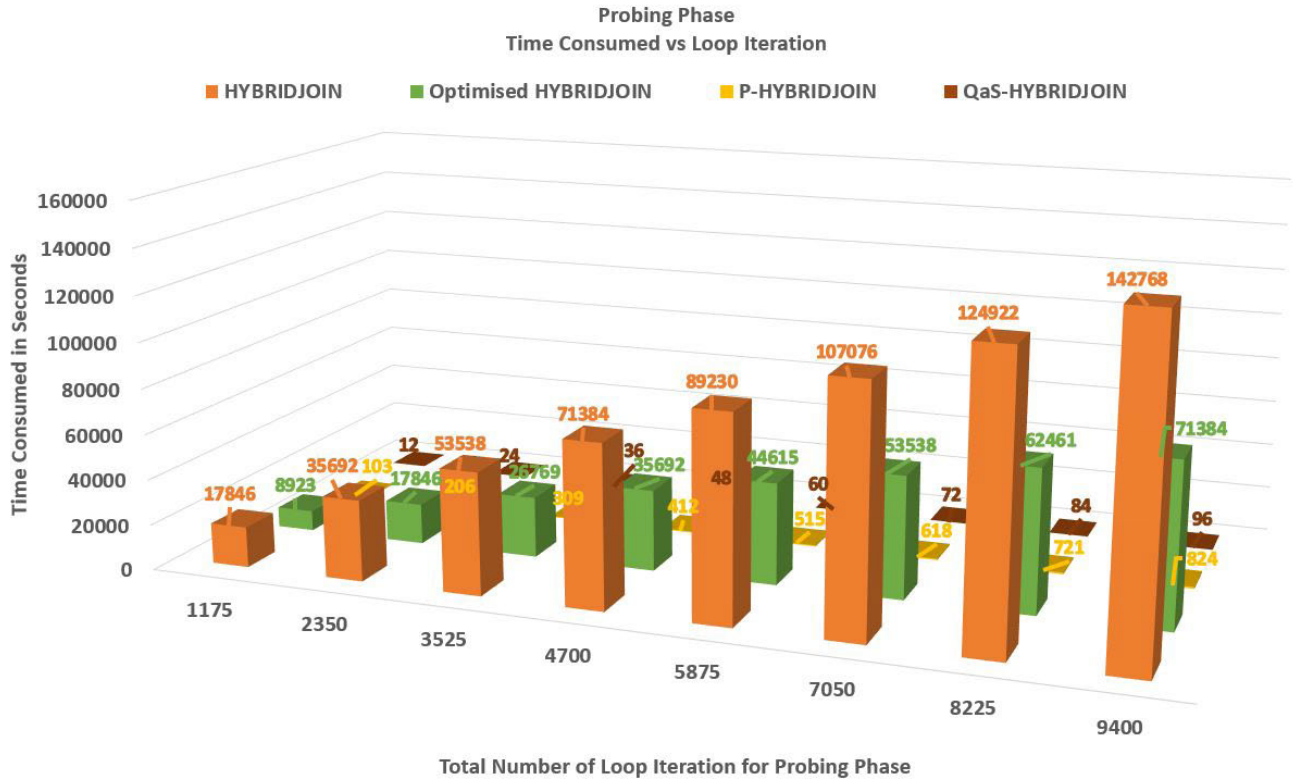

FIGURE 10. Disk I/O Cost Comparison of Improved Algorithms with Two Existing Variants(2 million tuples of R).

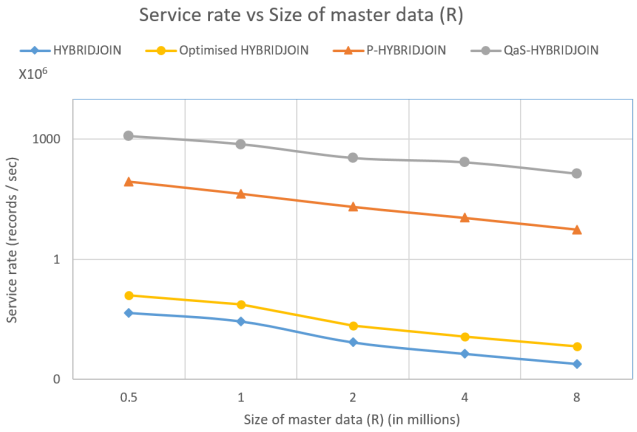

FIGURE 11. Service Rate vs Master Data.

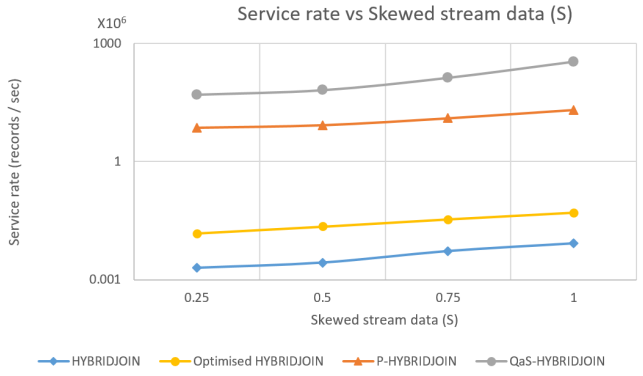

FIGURE 12. Service Rate vs Skewed Stream Data.

V. RESULTS AND DISCUSSION

Experiments performed using proposed P-HYBRIDJOIN and QaS-HYBRIDJOIN in terms of memory and processing show significant improvement as compared to original HYBRIDJOIN. The following sections show the experimental setup and cost comparison of proposed algorithms with two existing variants: HYBRIDJOIN and Optimised-HYBRIDJOIN.

A. EXPERIMENTAL SETUP

Proposed algorithms have been tested using the following hardware and data specification.

1) HARDWARE SPECIFICATION

The experiments were performed on the Core i7 hp pavilion laptop with 16 GB DDR. The maximum memory we had allocated to our algorithm was 250 MB. JAVA programming language is used for implementing proposed algorithm. Some of the built-in functions of JAVA were used for calculating the processing time i-e nanoTime(). The hash table in JAVA did not allow to store multiple records for one key value. Therefore, we have used the Apache Multi-Hash-Map for this purpose and for parallel process execution JAVA threads were used.

Algorithm 3 QaS-HYBRIDJOIN

procedure DISK BUFFER LOADING WINDOW
  Input: A master data R based on oldest and recent indexes from Queue and Stack QaS
  Output: (db1) or (db2)
  Parameters: a partition of master data R
  Method:
  while true **do**
    if (db1) state is empty **then**
      Get: the oldest key attribute as index from QaS
      Mark: (db1) state to busy
      Load: partition p from R in (db1) on oldest key attribute
      Mark: (db1) state to full
    end if
    if (db2) state is empty **then**
      Get: the recent key attribute as index from QaS
      Mark: (db2) state to busy
      Load: partition p from R in (db2) on recent key attribute
      Mark: (db2) state to full
    end if
  end while
end procedure

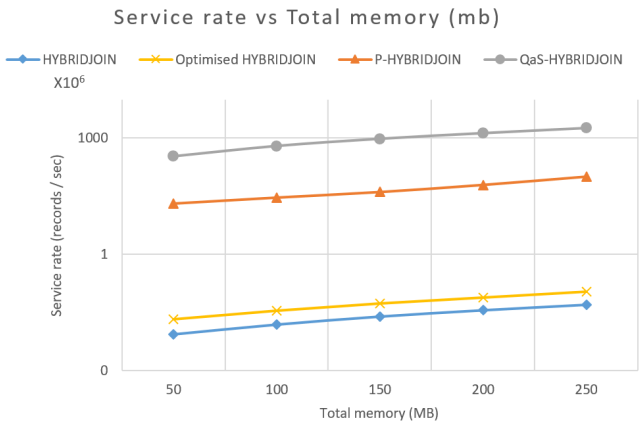

FIGURE 13. Service Rate vs Total Memory.

2) DATA SPECIFICATION

QaS-HYBRIDJOIN algorithm has been tested with various sizes for the master data R. Experiments have been executed with 0.5 million, 1 million, 2 million, 4 million, and 8 million records of master data R. Experiments used multiple memory allocation: 50MB, 100MB, 150MB, 200MB and 250MB of memory. Change in total memory alters the size for memory components. According to this setup, size of hash table H and QaS is directly proportional to the size of total memory. An increase in the size of total memory increases the size

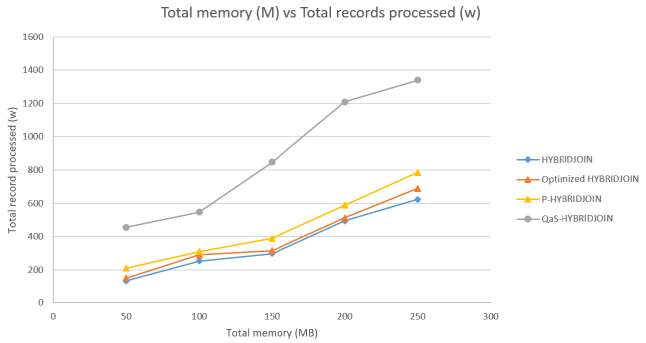

FIGURE 14. Processed Record vs Total Memory.

of QaS and H. However, memory buffers remain of fixed size of 1700 records where each record is of 120 Bytes. Stream generator designed for experiments generates random number of streams, also controlling delays. Component H has the fudge factor of 4.8 to handle skewed partitions.

3) SYSTEM OF MEASUREMENT

The performance of the algorithms has been measured by analyzing the number of records processed per second. This term is called service rate and denoted as μ. The results have been considered for evaluation only after execution of few loop iteration. For perfection of results we have taken 3 readings for each testing model and considered the third measurement for evaluation as model gets into flow by then and supposed to reflect more mature readings.

B. PERFORMANCE EVALUATION

We mainly evaluate the performance of proposed algorithms in terms of disk I/O cost, service rate, different sized disk/master data, skewed stream data, various sized memory budgets, numbers of matching records and cost validation(measured vs estimated).

1) DISK I/O COST

The major focus of this research is to improve the disk I/O cost of HYBRIDJOIN by developing new algorithms: P-HYBRIDJOIN and QaS-HYBRIDJOIN. The goal seemed to be achieved as proposed algorithms show significant reduced I/O cost as compared to original HYBRIDJOIN by introducing two disk buffers (db1, db2) and QaS as shown in Figure 10. Time consumed in seconds for disk I/O cost during various iterations have been measured for four mentioned algorithms. On total of 9400 iterations were recorded when algorithms were executed under 50 MB of total memory and 2 million records in master data R. Total eight readings were observed for each mentioned algorithm showing significant reduction in I/O cost in case of proposed algorithms as iterations grow. It can be further noted that proposed algorithms show consistent behaviour for growing iterations (unlimited execution of join window).

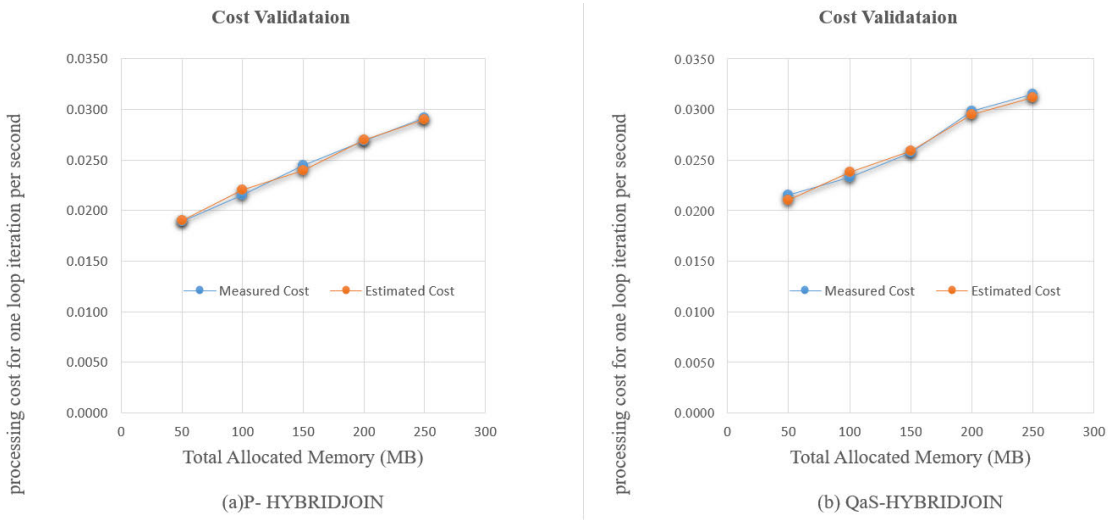

FIGURE 15. Cost Validation of Measured vs Estimated.

2) GROWING MASTER DATA

In our first experiment, we tested proposed algorithms for different sizes of master data R to achieve high service rate over growing master data R. Figure 11 plots the service rate over master data R for various sizes: 0.5 million tuples till 8 million tuples, twice the size pattern. Note that service rate tends to be decreased steadily when master data R size increased as search of required key takes longer time for bigger sized master data. Figure 11 clearly shows that P-HYBRIDJOIN performs better than existing variants of HYBRIDJOIN for all sizes of master data, whereas QaS-HYBRIDJOIN outperforms the rest.

3) SKEWED STREAM DATA

During second experiment we tested P-HYBRIDJOIN and QaS-HYBRIDJOIN, and varied the value of the Zipfian exponent e to evaluate stream distribution in real-life applications. Figure 12 shows that both improved algorithms perform significantly better than existing HYBRIDJOIN and Optimised-HYBRIDJOIN.

4) VARIOUS MEMORY BUDGETS

Figure 13 plots the service rate for different memory budgets: 50MB till 250MB with intervals of 50MB. This experiment was conducted to study the behaviour of improved algorithms under limited memory. Note that service rate keeps improving if memory is increased. Figure 13 zooms in the details of differences of HYBRIDJOIN, Optimized-HYBRIDJOIN, P-HYBRIDJOIN and QaS-HYBRIDJOIN.

5) NUMBER OF PROCESSED/MATCHING RECORDS

Figure 14 compares the number of processed records during execution of four mentioned algorithms for different memory budgets. This Figure clearly shows that QaS-HYBRIDJOIN processed more number of records as compared to other variants of HYBRIDJOIN. Additionally, it can be observed that number of processed streams will be significantly increased if more memory is reserved for components of proposed algorithms.

6) COST VALIDATION

In order to validate the performance of improved algorithms, we performed cost validation by comparing the measured and experimental costs of both algorithms. In the case of the predicted cost, we first determined the cost for one loop iteration. Figure 15 presents the comparisons of both costs for improved algorithms. In this Figure it can be observed that the predicted cost closely resembles the measured cost which validates the correctness of our implementations.

7) SUMMARY

Above experiments indicate that our two improved algorithms P-HYBRIDJOIN and QaS-HYBRIDJOIN exhibit better service rate and more number of processed records in a unit time under limited resources.

VI. CONCLUSION

The key contribution of this research is in the field of ETL for accessing disk based master data in an efficient way to improve the service rate by decreasing the I/O cost. We proposed two new improved HYBRIDJOIN algorithms: P-HYBRIDJOIN and QaS-HYBRIDJOIN that have decreased the I/O cost and increased the number of processed records. Initially, we have made the process of join and disk loading parallel to each other. Due to this approach, interdependent processes of disk buffer loading phase and probing phase are no more dependent on each other. By introducing two disk buffers for loading of disk partitions significantly reduced disk I/O cost. Consideration of oldest key attribute for join operation may reduce the number of matches in

existing variants of HYBRIDJOIN. This limitation has been addressed during second contribution of this study. Involvement of recent key attribute parallel to oldest key attribute for the join operation helped finding more matches. Therefore we introduced a new component queue and stack (QaS) by replacing the existing component queue Q. This new component has increased the performance by three times as compared to the original HYBRIDJOIN. We have made the implementations of all algorithms and evaluations publicly available to facilitate reproducible comparisons and further investigation of semi-stream join algorithms.

VII. FUTURE DIRECTION

This article has focused the equijoin for two different data sources which can also be implemented on non-equijoin scenarios like Internet of Things (IoT), smart devices,non uniform data streams and blockchain data sources. Moreover, proposed solutions can be used for images data streams for photogrammetry images of archaeology, defense, and Unnamed Ariel Vehicles (UAVs) for coastal changes. Proposed approaches can also be used for mulit-valued attributes data like motor cycle spare parts with different models, size, weight, and quality. The most of the join algorithms worked with text data, however, an images data streams of UAVs can be considered in future research with limited memory architecture for depth map generation and 3D Models.

REFERENCES

[1] J. Curran, F. Esser, D. C. Hallin, K. Hayashi, and C.-C. Lee, "International news and global integration: A five-nation reappraisal," *Journalism Stud.*, vol. 18, no. 2, pp. 118–134, Feb. 2017.
[2] K. Kasemsap, "Mastering business process management and business intelligence in global business," in *Organizational Productivity and Performance Measurements Using Predictive Modeling and Analytics*. Hershey, PA, USA: IGI Global, 2017, pp. 192–212.
[3] M. Vasudevan, Y.-C. Tian, M. Tang, and E. Kozan, "Profile-based application assignment for greener and more energy-efficient data centers," *Future Gener. Comput. Syst.*, vol. 67, pp. 94–108, Feb. 2017.
[4] O. Aziz, M. S. Farooq, A. Abid, R. Saher, and N. Aslam, "Research trends in enterprise service bus (ESB) applications: A systematic mapping study," *IEEE Access*, vol. 8, pp. 31180–31197, 2020.
[5] H. Hemingway, G. S. Feder, N. K. Fitzpatrick, S. Denaxas, A. D. Shah, and A. D. Timmis, "Using nationwide 'big data' from linked electronic health records to help improve outcomes in cardiovascular diseases: 33 studies using methods from epidemiology, informatics, economics and social science in the clinical disease research using linked bespoke studies and electronic health records (caliber) programme," Programme Grants Appl. Res. J., London, U.K., Tech. Rep. 4, 2017.
[6] L. Karim, A. Boulmakoul, and A. Lbath, "Near real-time big data analytics for NFC-enabled logistics trajectories," in *Proc. 3rd Int. Conf. Logistics Operations Manage. (GOL)*, May 2016, pp. 1–7.
[7] S. Cohen and W. H. Money, "Data systems fault coping for real-time big data analytics required architectural crucibles," in *Proc. 50th Hawaii Int. Conf. Syst. Sci.*, 2017, pp. 1–10.
[8] U. Sultan, "Real time data warehousing," Ph.D. dissertation, Dept. Comput. Sci., Inst. Bus. Admin. (IBA), Karachi, Pakistan, 2016.
[9] S. Ijaz Ahmad Bukhari, "Real time data warehouse," 2013, *arXiv:1310.5254*. [Online]. Available: http://arxiv.org/abs/1310.5254
[10] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*. San Mateo, CA, USA: Morgan Kaufmann, 2016.
[11] W. Fan and A. Bifet, "Mining big data: Current status, and forecast to the future," *ACM SIGKDD Explor. Newslett.*, vol. 14, no. 2, pp. 1–5, Apr. 2013.
[12] S. B. Eom, S. M. Lee, E. Kim, and C. Somarajan, "A survey of decision support system applications (1988–1994)," *J. Oper. Res. Soc.*, vol. 49, no. 2, pp. 109–120, 1998.
[13] R. Wodak, "'We have the character of an island nation'. A discourse-historical analysis of David cameron's' Bloomberg speech'on the European union," in *Browser Download This Paper*. Badia Fiesolana, Italy: European University Institute, 2016.
[14] E. B. Hagai and F. J. Crosby, "Between relative deprivation and entitlement: An historical analysis of the battle for same-sex marriage in the United States," in *Handbook of Social Justice Theory and Research*. Berlin, Germany: Springer, 2016, pp. 477–489.
[15] P. Vassiliadis and A. Simitsis, "Near real time ETL," in *New Trends in Data Warehousing and Data Analysis*. Berlin, Germany: Springer, 2009, pp. 1–31.
[16] G. D. Saxton, "CSR, big data, and accounting: Firms' use of social media for CSR-focused reporting, accountability, and reputation gain," York Space, North York, ON, Canada, Tech. Rep., 2016.
[17] G. Artač, B. Kladnik, D. Dovžan, M. Pantoš, and A. F. Gubina, "Demand-side system reserve provision in a stochastic market model," *Energy Sour., B, Econ., Planning, Policy*, vol. 11, no. 5, pp. 436–442, May 2016.
[18] J. Lowe-Power, M. D. Hill, and D. A. Wood, "When to use 3D die-stacked memory for bandwidth-constrained big data workloads," 2016, *arXiv:1608.07485*. [Online]. Available: http://arxiv.org/abs/1608.07485
[19] M. Lytras, V. Raghavan, and E. Damiani, "Big data and data analytics research: From metaphors to value space for collective wisdom in human decision making and smart machines," *Int. J. Semantic Web Inf. Syst.*, vol. 13, no. 1, pp. 1–10, 2017.
[20] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. New York, NY, USA: Manning Publications, 2015.
[21] J. Vidal-García, M. Vidal, and R. H. Barros, "Computational business intelligence, big data, and their role in business decisions in the age of the Internet of Things," in *The Internet of Things in the Modern Business Environment*. Hershey, PA, USA: IGI Global, 2017, pp. 249–268.
[22] W. Höpken and M. Fuchs, "Introduction: Special issue on business intelligence and big data in the travel and tourism domain," *Inf. Technol. Tourism*, vol. 16, no. 1, pp. 1–4, Mar. 2016.
[23] D. Chen, Y. Chen, B. N. Brownlow, P. P. Kanjamala, C. A. G. Arredondo, B. L. Radspinner, and M. A. Raveling, "Real-time or near Real-time persisting daily healthcare data into HDFS and ElasticSearch index inside a big data platform," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 595–606, Apr. 2017.
[24] E. Begoli, T. Dunning, and C. Frasure, "Real-time discovery services over large, heterogeneous and complex healthcare datasets using schema-less, column-oriented methods," in *Proc. IEEE 2nd Int. Conf. Big Data Comput. Service Appl. (BigDataService)*, Mar. 2016, pp. 257–264.
[25] C. Dussauge-Peisser, A. Helmstetter, J.-R. Grasso, D. Hantz, P. Desvarreux, M. Jeannin, and A. Giraud, "Probabilistic approach to rock fall hazard assessment: Potential of historical data analysis," *Natural Hazards Earth Syst. Sci.*, vol. 2, nos. 1–2, pp. 15–26, Jun. 2002.
[26] A. Bauer and M. Huber, "Near real-time modeling of pollution dispersion," U.S. Patent 20 170 091 350, Mar. 30, 2017.
[27] Y. Wang, L. Kung, W. Y. C. Wang, and C. G. Cegielski, "An integrated big data analytics-enabled transformation model: Application to health care," *Inf. Manage.*, vol. 55, no. 1, pp. 64–79, Jan. 2018.
[28] R. J. Santos and J. Bernardino, "Real-time data warehouse loading methodology," in *Proc. Int. Symp. Database Eng. Appl. IDEAS*, 2008, pp. 49–58.
[29] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "Stream: The stanford data stream management system," in *Data Stream Management*. Berlin, Germany: Springer, 2016, pp. 317–336.
[30] M. Garofalakis, J. Gehrke, and R. Rastogi, *Data Stream Management: Processing High-Speed Data Streams*. Berlin, Germany: Springer, 2016.
[31] B. Stryzak and T. E. Terwilliger, "System and method for multiple data channel transfer using a single data stream," U.S. Patent 14 693 575, Apr. 22, 2015.
[32] N. Kaur and S. K. Sood, "Efficient resource management system based on 4 Vs of big data streams," *Big Data Res.*, vol. 9, pp. 98–106, Sep. 2017.
[33] M. A. Naeem, G. Dobbie, and G. Weber, "Efficient processing of stream data over persistent data," in *Big Data Computing*. Boca Raton, FL, USA: CRC Press, 2013, p. 315.
[34] M. A. Naeem, G. Dobbie, G. Weber, and I. S. Bajwa, "Efficient usage of memory resources in near-real-time data warehousing," in *Proc. Int. Multi Topic Conf.* Berlin, Germany: Springer, 2012, pp. 326–337.
[35] D. Puthal, S. Nepal, R. Ranjan, and J. Chen, "A dynamic prime number based efficient security mechanism for big sensing data streams," *J. Comput. Syst. Sci.*, vol. 83, no. 1, pp. 22–42, Feb. 2017.

[36] Y. Im, W.-O. Oh, and M. Suk, "Risk factors for suicide ideation among adolescents: Five-year national data analysis," *Arch. Psychiatric Nursing*, vol. 31, no. 3, pp. 282–286, Jun. 2017.
[37] Y. Joti, K. Nakajima, T. Kameshima, M. Yamaga, T. Abe, K. Okada, T. Sugimoto, T. Hatsui, and M. Yabashi, "Data analysis environment for X-ray free-electron laser experiments at SACLA," *Synchrotron Radiat. News*, vol. 30, no. 1, pp. 16–21, Jan. 2017.
[38] C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk, and O. Spatscheck, "Gigascope: High performance network monitoring with an SQL interface," in *Proc. ACM SIGMOD Int. Conf. Manage. Data SIGMOD*, 2002, p. 623.
[39] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Quicksand: Quick summary and analysis of network data," Citeseer, Princeton, NJ, USA, Tech. Rep. 2001-43, Dec. 2001. [Online]. Available: nj.nec.com/gilbert01quicksand.html
[40] S. Madden and M. J. Franklin, "Fjording the stream: An architecture for queries over streaming sensor data," in *Proc. 18th Int. Conf. Data Eng.*, Feb. 2002, pp. 555–566.
[41] M. Sullivan and A. Heybey, "A system for managing large databases of network traffic," in *Proc. USENIX*, 1998, p. 2.
[42] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts, "Linear road: A stream data management benchmark," in *Proc. 13th Int. Conf. Very Large Data Bases (VLDB)*, vol. 30, 2004, pp. 480–491.
[43] Y. Guo, M. Wang, and X. Li, "Application of an improved apriori algorithm in a mobile e-commerce recommendation system," *Ind. Manage. Data Syst.*, vol. 117, no. 2, pp. 287–303, Mar. 2017.
[44] A. Arasu, S. Babu, and J. Widom, "An abstract semantics and concrete language for continuous queries over streams and relations," Stanford Univ., Stanford, CA, USA, Tech. Rep., 2002.
[45] Z. Wang, Z. Wang, S. He, X. Gu, and Z. F. Yan, "Fault detection and diagnosis of chillers using Bayesian network merged distance rejection and multi-source non-sensor information," *Appl. Energy*, vol. 188, pp. 200–214, Feb. 2017.
[46] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *Proc. Int. Conf. Mobile Data Manage.* Berlin, Germany: Springer, 2001, pp. 3–14.
[47] C. Lee and C. Lee, "Method to reduce the gap between construction and IT companies to improve suitability before selecting an enterprise system," *Comput. Ind.*, vol. 85, pp. 23–30, Feb. 2017.
[48] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong, "Design considerations for high fan-in systems: The HiFi approach," in *Proc. CIDR*, vol. 5, 2005, pp. 24–27.
[49] H. Gonzalez, J. Han, X. Li, and D. Klabjan, "Warehousing and analyzing massive RFID data sets," in *Proc. 22nd Int. Conf. Data Eng. (ICDE)*, Apr. 2006, p. 83.
[50] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in *Proc. ACM SIGMOD Int. Conf. Manage. Data - SIGMOD*, 2006, pp. 407–418.
[51] R. M. Monczka, R. B. Handfield, L. C. Giunipero, and J. L. Patterson, *Purchasing and Supply Chain Management*. Boston, MA, USA: Cengage Learning, 2015.
[52] S. Sudha and S. Manikandan, "M-hybridjoin-an adaptive approach for stream based near real-time data warehousing," *Int. J. Adv. Eng. Tech.*, vol. 321, p. 326, Jan. 2016.
[53] N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, and N.-E. Frantzell, "Supporting streaming updates in an active data warehouse," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, Apr. 2007, pp. 476–485.
[54] N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, and N. Frantzell, "Meshing streaming updates with persistent data in an active data warehouse," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 7, pp. 976–991, Jul. 2008.
[55] A. N. Wilschut and P. M. Apers, "Pipelining in query executions," IEEE, New York, NY, USA, Tech. Rep., 1990.
[56] A. N. Wilschut and P. M. G. Apers, "Dataflow query execution in a parallel main-memory environment," *Distrib. Parallel Databases*, vol. 1, no. 1, pp. 103–128, Jan. 1993.
[57] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, and D. S. Weld, "An adaptive query execution system for data integration," in *Proc. ACM SIGMOD Int. Conf. Manage. Data - SIGMOD*, 1999, pp. 299–310.
[58] T. Urhan and M. Franklin, "XJoin: A reactively-scheduled pipelined join operator," *IEEE Data Eng. Bull.*, vol. 23, pp. 27–33, 2000.
[59] T. Urhan and M. J. Franklin, "Xjoin: Getting fast answers from slow and bursty networks," Digit. Repository Univ. Maryland, College Park, MD, USA, Tech. Rep., 1999.
[60] T. Urhan and M. J. Franklin, "Dynamic pipeline scheduling for improving interactive query performance," in *Proc. VLDB*, vol. 1, 2001, pp. 501–510.
[61] M. F. Mokbel, M. Lu, and W. G. Aref, "Hash-merge join: A non-blocking join algorithm for producing fast and early join results," in *Proc. 20th Int. Conf. Data Eng.*, Apr. 2004, pp. 251–262.
[62] R. Lawrence, "Early hash join: A configurable algorithm for the efficient and early production of join results," in *Proc. 31st Int. Conf. Very Large Data Bases*, Aug. 2005, pp. 841–852.
[63] M. A. Hammad, W. G. Aref, and A. K. Elmagarmid, "Stream window join: Tracking moving objects in sensor-network databases," in *Proc. 15th Int. Conf. Sci. Stat. Database Manage.*, Jul. 2003, pp. 75–84.
[64] M. Golfarelli, S. Rizzi, and I. Cella, "Beyond data warehousing: What's next in business intelligence?" in *Proc. 7th ACM Int. Workshop Data Warehousing OLAP - DOLAP*, 2004, pp. 1–6.
[65] M. A. Naeem, G. Dobbie, G. Weber, and S. Alam, "R-MESHJOIN for near-real-time data warehousing," in *Proc. ACM 13th Int. Workshop Data Warehousing OLAP - DOLAP*, 2010, pp. 53–60.
[66] R. Ramakrishnan and J. Gehrke, *Database Management Systems*. New York, NY, USA: McGraw-Hill, 2000.
[67] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, "On supporting containment queries in relational database management systems," in *Proc. ACM SIGMOD Int. Conf. Manage. Data - SIGMOD*, 2001, pp. 425–436.
[68] D. J. DeWitt, J. F. Naughton, and J. Burger, "Nested loops revisited," in *Proc. 2nd Int. Conf. Parallel Distrib. Inf. Syst.*, Jan. 1993, pp. 230–242.
[69] D. J. DeWitt, J. F. Naughton, and D. A. Schneider, "An evaluation of non-equijoin algorithms," in *Proc. 17th Int. Conf. Very Large Data Bases*, San Mateo, CA, USA: Morgan Kaufmann, 1991, pp. 443–452.
[70] D. J. DeWitt, J. F. Naughton, and D. A. Schneider, "Parallel sorting on a shared-nothing architecture using probabilistic splitting," in *Proc. 1st Int. Conf. Parallel Distrib. Inf. Syst.*, 1991, pp. 280–291.
[71] A. Gupta and I. S. Mumick, "Maintenance of materialized views: Problems, techniques, and applications," *IEEE Data Eng. Bull.*, vol. 18, no. 2, pp. 3–18, Jun. 1995.
[72] W. Kim, "A new way to compute the product and join of relations," in *Proc. ACM SIGMOD Int. Conf. Manage. Data - SIGMOD*, 1980, pp. 179–187.
[73] W. J. Labio, J. L. Wiener, H. Garcia-Molina, and V. Gorelik, "Efficient resumption of interrupted warehouse loads," in *Proc. ACM SIGMOD Int. Conf. Manage. Data - SIGMOD*, 2000, pp. 46–57.
[74] L. D. Shapiro, "Join processing in database systems with large main memories," *ACM Trans. Database Syst.*, vol. 11, no. 3, pp. 239–264, Aug. 1986.
[75] E. J. Shekita and M. J. Carey, *A Performance Evaluation of Pointer-Based Joins*, vol. 19. New York, NY, USA: ACM, 1990.
[76] X. Zhang and E. A. Rundensteiner, "Integrating the maintenance and synchronization of data warehouses using a cooperative framework," *Inf. Syst.*, vol. 27, no. 4, pp. 219–243, Jun. 2002.
[77] Y. Zhuge, H. García-Molina, J. Hammer, and J. Widom, "View maintenance in a warehousing environment," *ACM SIGMOD Rec.*, vol. 24, no. 2, pp. 316–327, May 1995.
[78] K. Bratbergsengen, "Hashing methods and relational algebra operations," in *Proc. 10th Int. Conf. Very Large Data Bases*, San Mateo, CA, USA: Morgan Kaufmann, 1984, pp. 323–333.
[79] H. Gao, M. A. Naeem, C. Lutteroth, and G. Weber, "S3J: A parallel semi-stream similarity join," in *Proc. ACM 18th Int. Workshop Data Warehousing OLAP*, Oct. 2015, pp. 49–57.
[80] M. A. Bornea, A. Deligiannakis, Y. Kotidis, and V. Vassalos, "Semi-streamed index join for near-real time execution of ETL transformations," in *Proc. IEEE 27th Int. Conf. Data Eng.*, Apr. 2011, pp. 159–170.
[81] D. R. Oran, "Retransmission-based stream repair and stream join," U.S. Patent 9 083 585, Jul. 14, 2015.
[82] M. A. Naeem, G. Dobbie, and G. Weber, "A lightweight stream-based join with limited resource consumption," in *Proc. Int. Conf. Data Warehousing Knowl. Discovery*. Berlin, Germany: Springer, 2012, pp. 431–442.
[83] T. M. Tran and B. S. Lee, "Distributed stream join query processing with semijoins," *Distrib. Parallel Databases*, vol. 27, no. 3, pp. 211–254, Jun. 2010.
[84] A. Al-Badarneh, M. Al-Rudaini, F. Ali, and H. Najadat, *Index-Based Join in Mapreduce Using Hadoop Mapfiles*. Jaipur, India: Global Research & Development Services, 2016.
[85] A. Chakraborty and A. Singh, "A partition-based approach to support streaming updates over persistent data in an active datawarehouse," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–11.
[86] H. Roh, M. Shin, W. Jung, and S. Park, "Advanced block nested loop join for extending SSD lifetime," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 4, pp. 743–756, Apr. 2017.

[87] J. Fang, X. Wang, R. Zhang, and A. Zhou, "Flexible and adaptive stream join algorithm," in *Proc. Asia–Pacific Web Conf.* Berlin, Germany: Springer, 2016, pp. 3–16.
[88] J. Fang, R. Zhang, X. Wang, T. Z. J. Fu, Z. Zhang, and A. Zhou, "Cost-effective stream join algorithm on cloud system," in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2016, pp. 1773–1782.
[89] M. A. Naeem, G. Dobbie, and G. Weber, *Hybridjoin for Near-Real-Time Data Warehousing*. Hershey, PA, USA: IGI Publishers, 2011.
[90] M. A. Naeem, G. Dobbie, and G. Weber, "X-hybridjoin for near-real-time data warehousing," in *Proc. Brit. Nat. Conf. Databases*. Berlin, Germany: Springer, 2011, pp. 33–47.
[91] M. A. Naeem, G. Dobbie, and G. Weber, "Optimised x-hybridjoin for near-real-time data warehousing," in *Proc. 23rd Australas. Database Conf.*, vol. 124. Darlinghurst, NSW, Australia: Australian Computer Society, Inc., 2012, pp. 21–30.
[92] M. A. Naeem, "Tuned x-hybridjoin for near-real-time data warehousing," in *Proc. Asia–Pacific Web Conf.* Berlin, Germany: Springer, 2013, pp. 494–505.
[93] M. A. Naeem, O. Aziz, and N. Jamil, *Optimising Hybridjoin to Process Semi-Stream Data in Near-Real-Time Data Warehousing*. Berlin, Germany: Springer, 2019.

OMER AZIZ received the M.S. degree in computer science from the National College of Business Administration and Economics, Lahore, Pakistan. He is currently pursuing the Ph.D. degree in computer science with the University of Management and Technology, Lahore. He is currently working as a Lecturer with the Department of Computer Science, NFC Institute of Engineering and Technology, Multan. He has 14 years of professional experience in education and industry. He developed software applications, websites, and mobile application for different companies around the globe. He has strong analysis and software architecture design skills according to emerging software market demand of data science, machine learning, cross platform, and artificial intelligence.

TAYYABA ANEES was born in Pakistan. She received the Ph.D. degree from the Vienna University of Technology, Vienna, Austria, in 2012. Her Ph.D. dissertation is in the area of service-oriented architecture and web services availability domain. She has worked as a Project Assistant with the Vienna University of Technology for four years. She is currently working as the Program Head Software Engineering/an Assistant Professor with the Department of Software Engineering, School of Systems and Technology, University of Management and Technology, Lahore. Her research interests include service-oriented architecture, Web services, Web of Things (WOT), Semantic Web, software availability, software safety, software fault tolerance, and real-time data warehousing.

ERUM MEHMOOD was born in Pakistan. She received the M.Phil. degree in computer science from the National College of Business Administration and Economics (NCBAE), Lahore, Pakistan, in 2017. She is currently pursuing the Ph.D. degree with the University of Management and Technology, Lahore. Her M.Phil. dissertation is in the area of stream processing for real-time data warehousing.

She is currently working as a Lecturer of Computer Science with Government Degree College, Lahore. Her research interests include big data analytics, stream processing, ETL, and real-time data warehousing.