IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 7, 2019, accepted September 6, 2019, date of publication September 17, 2019,
date of current version September 27, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2941925

Performance Analysis of Not Only SQL Semi-Stream Join Using MongoDB for Real-Time Data Warehousing

ERUM MEHMOOD AND TAYYABA ANEES
School of Systems and Technology, University of Management and Technology, Lahore 54770, Pakistan
Corresponding author: Erum Mehmood (erum9964@hotmail.com)

ABSTRACT Data warehousing has been indispensable to enterprises for decades. However, infrequently updated data warehouse environment does not support quicker business decisions and faster data recovery in case of transformation or load issue. Implementation of real-time data warehouse provides solution to update problems of enterprises. Efficient stream processing for un-structured(NoSQL) and structured(SQL) data from various sources is required for the successful implementation of real-time data warehousing. We have done an analysis between un-structured and structured semi-stream join processing, using efficient database engine MongoDB at Extraction-Transformation-Loading phase. Semi-stream tuples coming from different sources are joined with disk-based master data, based on keys in memory, for both un-structured and structured documents(tuples) using MongoDB server, where the I/O rates are different for both inputs. Through experiments, in this paper we have analyzed the CPU and memory usage for real-time semi-stream join processing through two types of tests, un-structured and structured data streams using synthetic and real datasets. The results show that, memory usage and execution time remains consistent for a given specification irrespective of the nature of data streams (un-structured or structured), even when incoming semi-streams are growing.

INDEX TERMS NoSQL/SQL, semi-stream join, real-time data warehousing, MongoDB.

I. INTRODUCTION

For deriving intelligence out of data, need of data warehouse (DW) is increasing nowadays. Rather than having multiple decision-support environments operating independently, which may lead to conflicting information, a DW unifies all sources of information. Decisional support system architecture is composed of three important phases: DW building, exportation and Extracting, Transforming and Loading (ETL) processes which are responsible for extracting, transforming and loading data into a multidimensional DW [1] as shown in figure 1. In order to build ETL for traditional DW, structure of the target system needs to be known in advance. As NoSQL databases are schema-free, this increases the need for extending the existing ETL tool in order to be able to designing schema while integrating data.

Basic purpose of ETL is to filter redundant data not required for analytical reports and to converge data for fast report generation [2]. Online Transaction Processing (OLTP) refers to workloads that access data randomly, typically to perform a quick search, insert, update or delete. OLTP operations are normally performed concurrently by a large number of users who use the database in their daily work for regular business operations. Typical data sources for a DW for Enterprise Resource Planning (ERP) systems are from external sources as well as from internal sources. Large data files are required to hold heterogeneous data temporarily which is extracted from multiple operational systems, called streams, after regular prescribed intervals. Extracted data elements are moved to staging area after reformatting and rearranging the data streams [3].

The idea of non-relational (NoSQL) databases alludes to a database option in contrast to the relational model that organizes information discretely into tables of rows and columns. Whereas NoSQL databases organize information into document-based database, key-value store etc. The document-based databases allow the storage of documents made up of tagged elements and they seem to

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

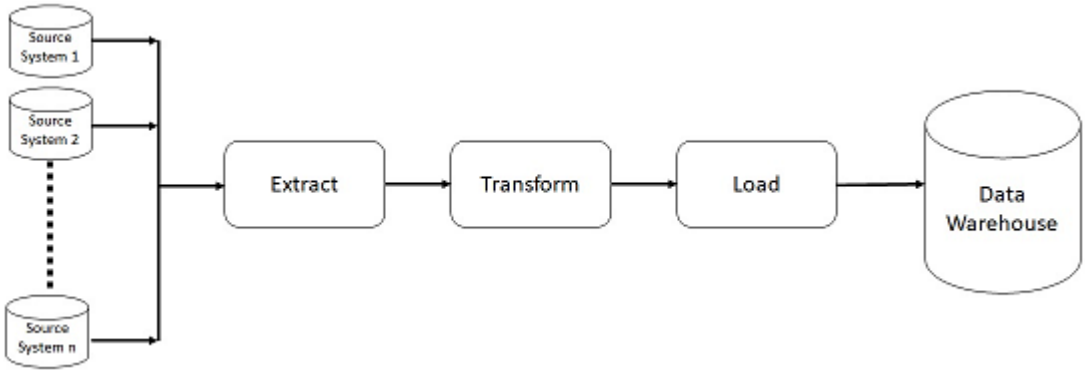

FIGURE 1. ETL phase for streams coming from various sources.

be the most popular NoSQL databases in practice (examples: Couchbase, MongoDB [4]; also in Latvia developed Clusterpoint [5]). While key-value stores use big hash tables of keys and values for fast accessing of data (examples: Riak, Amazon Dynamo) [6]. MongoDB is a NoSQL database which contains collections of documents which has no schema. A set of key/value pairs is designated as a document, and these documents are associated with dynamic schemas making NoSQL databases as high performance data structures which are suitable for lookup and filtering operations therefore used for the implementation of our work.

Today's most common DW implementation is based on the relational model using SQL as its query language. However, Not Only SQL (NoSQL) DW solutions are being proposed by many researchers as they are more scalable and have better performance in comparison to relational data bases. Instances of this type of data are smart phone records in which the location is broadcast in short and regular intervals, videos from cameras in public areas and even the extensive number of documents on the web. Complex join queries are required for selecting and joining information from multiple tables in relational DW, which take longer time for large schema. Structured (SQL) vs Un-structured (NoSQL) database terminology comparison is shown in table

Semi-structured data from XML and web pages may not be represented by any kind of schema [7] where similar data objects may have different characteristics. However, NoSQL DW can help deal with such data. A few researches have been done in the area of ETL phase in NoSQL warehousing [1], [8] due to its complexity as NoSQL databases are schema-free.

According to the demand of growing business industry, data streams immediately need to be loaded into DW almost at real-time pace. However, format of data streams from multiple sources is schema free and sometimes undefined which leads in need of document oriented (NoSQL) real-time DW instead of Relational DBMS based DW.

In this paper, we present performance analysis of Not Only SQL semi-stream join for real-time data warehousing. For this analysis we implement semi-stream join process for

TABLE 1. Structured vs Un-structured database terminologies [9].

Description	Structured	un-structured
Data structure	All of your data must be in same well defined structure assuming the attributes of this data are defined in advance.	Not necessary to define attributes of data or data structure in advance, a flexible structure.
Relationship between tables database	Very well organized and fully referenced	Relationship between collections is not necessary.
Transaction	Use ACID: Atomicity Consistency Isolation Durability	Use of CAP Theorem (Consistency, Availability, Partition) tolerance which expresses that all these characteristics are impossible to achieve simultaneously.
Indexes and queries	SQL query	Reduced the use of indexing and SQL as standard query

NoSQL streams of data in real-time DW scenario: discussed in section III, which extracts data streams from two different sources and join them after rearranging and applying filter on real-time basis according to the format of DW. We have run experiments using big-sized NoSQL(synthetic and real-life) and SQL databases in order to analyze the performance of semi-stream join process. We have compared performance aspect of our work in terms of different disk-based master data sizes and memory budgets. Outcomes from our experiments give insight into feasibility of using a NoSQL database for semi-stream join algorithms. Our motivation for this study is to analyze the performance of semi-stream join process in case when data streams are schema free and disk-data is huge in size and to understand and evaluate their memory usage and processing speed.

This section describes the purpose of real-time DW and the need to capture and evaluate huge sized un-structured schema free data streams on real-time basis for business intelligence.

The remainder of this paper is organized as follows: Related work is presented in section II. In section III, NoSQL

semi-stream join process using MongoDB for real-time DW is described in detail. In section IV, experimental evaluation using synthetic and real-life datasets is discussed, pointing features of memory distribution and disk-based master data size. Conclusion and future work appear in Section V.

II. RELATED WORK

Due to the importance of ''Big Data'' NoSQL databases are gaining popularity. Performance of some NoSQL and SQL databases are investigated independently by [10] in the light of key/value stores and result indicates that not all NoSQL databases perform better than SQL databases. For each database, the performance varies with each read or write operation and there is little correlation between performance and the data model each database uses. However, this discussion can not provide any solution for NoSQL stream joins for real-time DW.

Response time on relational and non-relational database models is focused in the work by [9], where several load tests are performed on SQL Server and MongoDB. MongoDB offers better performance over Linux operating systems, while SQL Server tends to increase response time when it comes to quantity. However, SQL/NoSQL stream join operation using SQL and NoSQL databases is not focused in this work.

The study by [7] analyzed the performance of the NoSQL database against the traditional SQL database for processing a modest amount of structured data by implementing databases in MongoDB and SQL server. The authors concluded that MongoDB performs equally as well or better than the relational database even for growing database, besides when aggregate functions are deployed. However, this study does not compare MongoDB implementation of stream join operation for real-time DW with SQL implementation.

Comparison of different database engines is presented in [11]. Speed analysis of HBase, Cassandra and MySQL on different workloads is experimented in the mentioned research. The experiments show that HBase performs better for Read, Update and Insert operations. However, we choose MongoDB and MongoDB Atlas for the experiments of our research because of its key-value store structure and as a fully automated cloud service.

A new approach to adapt Extract-Transform-Load (ETL) processes with Big Data technologies called Big Dimensional ETL (BigDimETL) is presented by [8] that deals with ETL development process and taking into account the Multidimensional structure. Experimental results show that ETL operation adaptation can perform well especially with Join operation using this approach.

Typical RDBMS are inefficient for handling un-structured big data generated by Web, in contrast NoSQL have the capabilities of handling such kind of data described by [6] and [1]. The aim of the research in [12] on NoSQL ETL is vital since currently DWs consist of structured data and this does not lead to storage of un-structured data in DW hence, strategic business decisions and data analysis can not be made for un-structured data. This study plays an important role in selection of join operation at ETL phase for un-structured data streams for real-time data warehousing.

The research by [6] is based on development and use of prototypes for NoSQL based DW. The authors concluded that creation of NoSQL based DW is possible only if a good balance is found between characteristics of SQL based DW and functionality offered by NoSQL DBMS. However, this study does not provide an insight into NoSQL stream joins at ETL phase for the implementation of NoSQL based DW.

According to [13], horizontally scalable stream processors are gaining momentum as the requirement for low latency has become a driving force in modern Big Data analytics pipelines. Performance analysis on very powerful stream processing systems, Apache Spark, Apache Smaza and Apache Storm, in terms of tuple sizes is presented by [14] and [13]. However, both studies have not addressed NoSQL streams for real-time DW. Large scale data analytics for real-time streaming sensory and social media data in smart city environments is presented in another research by [15], that provides the architecture to extract actionable-knowledge out of raw sensory data (NoSQL). This study, however, doesn't provide an insight into stream processing for real-time DW.

The MESHJOIN algorithm proposed by [16] used for matching a continuous real-time structured data stream with a big sized disk-based database. This algorithm plays a prominent role in the field of stream joins as it analyzes and joins inputs coming from two different sources: i.e., stream and disk. However, this algorithm cannot deal in un-structured input data stream for NoSQL based real-time DW.

A robust stream-based stream join algorithm, called HYBRIDJOIN presented in [17], deals with structured streams of data for real-time DW. Non-uniform distributions [18] of stream data are not implemented by the HYBRIDJOIN. However, both of these approaches have not addressed un-structured stream joins for real-time DW.

An adaptive semi-stream join algorithm called CACHEJOIN focusing nonuniform SQL stream data is presented in [19]. Another approach, P-CACHEJOIN (Parallel Cache Join) algorithm for semi-stream joins in real-time data warehousing is presented in [20] with some optimization in CACHEJOIN. However, these algorithms have not provided solution in case of un-structured stream joins for real-time DW.

A many-to-many semi-stream join (SSBJ) with a theoretically founded caching strategy is proposed by [21]. The cache in SSBJ does not interfere with the subsequent disk-based join, therefore it is possible to apply this caching strategy to other semi-stream joins in the future. However, SSBJ does not provide solution for un-structured stream joins for real-time DWH.

In the [22], the authors tested population-based queries of NoSQL databases storing archetype-based Electronic Health Record (EHR) data. Population-based queries were submitted to relational database, XML databases and NoSQL Couchbase. Authors concluded that Couchbase had better response times than MySQL, especially for larger datasets.

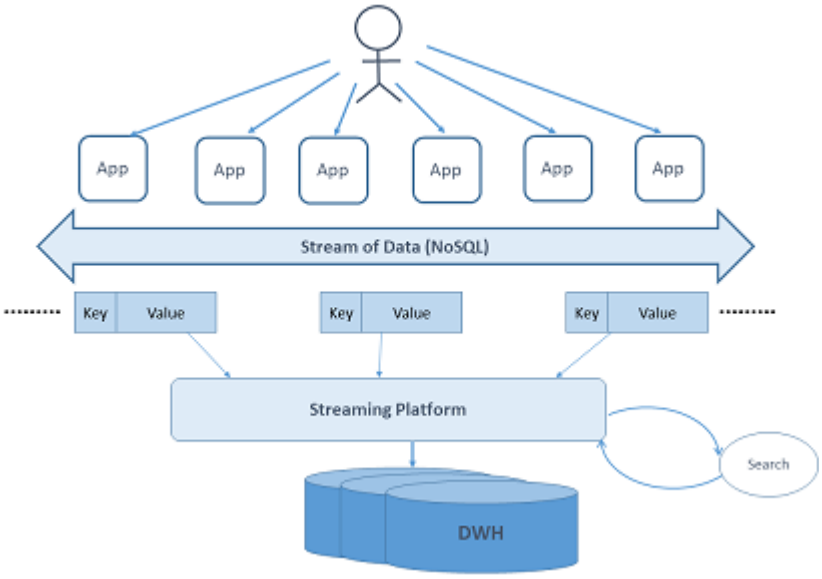

FIGURE 2. Real-time stream processing architecture for un-structured data from various apps.

Another study by [23] evaluated the scalability of PyEHR, a data access layer, using two NoSQL Database Management Systems: MongoDB and Elasticsearch, for secondary use of structured heterogeneous biomedical and clinical data. However, the un-structured aspect of data is not the focus of these studies.

As shown above, several approaches for analyzing the performance of SQL and NoSQL databases using structured or un-structured data have been presented. To the best of our knowledge, however, there is no such study in the literature that deals with un-structured semi-stream joins using MongoDB for real-time data warehousing.

III. METHOD AND MATERIAL

In this section, we propose implementation method named join module for the performance analysis of not only SQL semi-stream join for real-time DW at ETL layer. Architecture shown in figure 2, presents real-time stream processing at ETL layer through NoSQL semi-stream data in real-time DW environment, is followed for the implementation of this analysis. We analyzed the performance of SQL and NoSQL stream processing through join module in order to provide a performance comparison for SQL and NoSQL data. For stream processing implementation, we first generated six synthetic datasets each of different size and type and one real-life dataset. Secondly we generated six stream files for synthetic datasets and one stream file for real-life dataset containing ids from each dataset used for stream generation.

MongoDB Compass with MongoDB Server version 3.0 have been installed for our implementation on a machine running an Intel Core i5 1.70 GHz processor with 4GB of RAM. Proposed applications were coded in python using IDLE (python 3.6 64-bit).

Sub-section A explains testing arrangements for the execution of our implementation in order to conduct this analysis. Detailed data specifications for each dataset and stream file are presented in sub-section B whereas overall working of join module is described in sub-section C.

A. TESTING ARRANGEMENTS

We have used one, two and four million number of un-structured and structured documents as synthetic datasets, total are six, and one million number of un-structured documents from one real-life dataset to perform this analysis. Different sized data streams starting from ten till twenty thousand number of streams are tested in our join module. Results of join operation of ten, one hundred, five hundred, one thousand, ten thousand, fifteen thousand and twenty thousand number of streams from each stream file are included in this analysis as no significant results were generated for other sizes. In this analysis, we observed execution time and memory usage for stream processing only for the mentioned stream sizes as the results for streams more than maximum limit can be predicted on the basis of pattern identified from other tested sizes and are discussed in detail in section IV.

B. DATA SPECIFICATIONS

We used two categories of datasets that we refer to in the following as synthetic data and real-life data. We describe now the characteristics of these two categories.

1) SYNTHETIC DATA

For the purpose of evaluating performance of our implementation we created syntheic dataset similar to real-life datasets and at least as challenging if not more, for stream join process. According to proposed testing arrangement, we generated

TABLE 2. Data specifications for synthetic un-structured datasets and stream files.

	Un-Structured Collections		
	unstr-col-1	**unstr-col-2**	**unstr-col-3**
Num. Documents	1,000,000	2,000,000	4,000,000
Avg. Document Size	570.2 B	569.5 B	555.0 B
Total Document Size	543.8 MB	1.1 GB	2.1 GB
Num. Index	1	1	1
Total Index Size	9.5 MB	19.5 MB	38.5 MB
Stream File Name(.txt)	unstr-file-1	unstr-file-2	unstr-file-3
Stream File Size	26,368 KB	52,735 KB	105,469 KB

TABLE 3. Data specifications for synthetic structured datasets and stream files.

	Structured Collections		
	str-col-1	**str-col-2**	**str-col-3**
Num. Documents	1,000,000	2,000,000	4,000,000
Avg. Document Size	343.0 B	343.0 B	343.0 B
Total Document Size	327.1 MB	654.2 MB	1.3 GB
Num. Index	1	1	1
Total Index Size	9.5 MB	19.1 MB	38.4 MB
Stream File Name(.txt)	str-file-1	str-file-2	str-file-3
Stream File Size	25,391 KB	50,782 KB	101,563 KB

six un-structured and structured dataset collections starting from one million number of documents till four million number of documents. Sample collections for un-structured and structured data are stored on disk. Stream files containing ids from each generated dataset are also generated and stored on disk. Table 2 shows data specifications for un-structured collections containing thirty or less attributes with their corresponding stream file names and sizes. We have generated structured datasets each with thirty or less attributes of integer data type using MongoDB compass without creating any nested documents in order to maintain simple schema. Three collections were generated for structured data, data specification of each collection along with their stream file names and sizes are shown in table 3.

Three random documents from disk-based un-structured collection "unstr-col-1" are shown in table 4 representing nested documents each of different type and size. Documents with attributes of different data types are generated for experiments in order to create un-structured collection where id field (hexadecimal) is a primary key. Documents inside documents are generated for more complex schema, array and object fields indicate nested documents shown in second and third row of table 4.

2) REAL-LIFE DATA

In order to validate the results, we experimented semi-stream join process on real-life dataset deployed from cloud MongoDB Atlas SandBox cluster.[1]

[1] https://docs.atlas.mongodb.com/sample-data/sample-weather/

TABLE 4. Three random disk-based documents(tuples) from "unstr-col-1" collection.

No.	Fields	Data types	Values
1	12	id	5c51b5ad53a0811540b8c270 (PK)
		Int32	58272
		String	ZvXLkzDPujACDmMItn ZvyQBXfqt-zOkyMgvhKhIwZwZARGWOAx
		Int32	-22124
		String	PxUMihjh oaGKArneTMEEolomgPh
		Int32	-13
		Array	[29259,3299,3358,2201,4801,131, 3187,4029] (Nested Document)
		String	KyrfXsRSn23 QKIghg 7BFTQZtXh-TOAdEs
		Int32	101961
		Int32	-92839
		Double	11184.056608873256
		Double	-432.32
2	15	id	5c51b57753a0811540b825cf
		Double	7008.171277545948
		String	sdfgkj
		String	az1yobh3epkulrp3 iji
		Int32	56681
		Double	-1684.5541894
		Double	-323.21031726915317
		String	ydHmh rpuku234g DiumAsXmZqr
		String	dpyuo5y6eniz1flu4y9
		String	gkeolrkwertw
		Double	9732.880329
		String	kc9gaybuu7ghx0vr51f
		Int32	59191
		Double	-37114.2
		Array-Float	[17092.749393559054, 5317.8860047789, 22793.713487125282,10110.6] (Nested Document)
3	10	id	5c51abf353a0812490885bd8
		Double	7008.171277545948
		String	sdfgkj
		String	az1you4hbh3epkulrp3
		Object	[7852, [28846.48, 16997.7, 4712.4392854, 45361.69686096254], 9352.841646531975, 'LRvAxJSxlW'] (Nested document)
		Array-Float	[16254.832, 5969.06412250672, 8004.647178243394] (Nested Document
		Int32	56681
		String	2ydlorpvxq1uc1
		Int32	456
		Double	-1.34456

We have selected collection "100YWeatherSmall" to be used in our analysis from cloud MongoDB Atlas SandBox cluster. When any new weather condition analysis needs any search, a real-time analytics is required to match the query to its massive data stores.

The motivation of this data set is to analyze the performance of semi-stream join process on a standardized workload as this database is common in big data infrastructure and captures massive data in an un-structured format, and is also used in experiments by [24] in their micro-batch model for distributed join processing. In order to make this dataset fit on our proposed testing arrangements, we increased its size by duplicating whole dataset four times obtaining one million of documents. Initial and modified data specifications for real-life dataset "100YWeatherSmall" are presented in table 5. We present twenty fields of one random document from "100YWeatherSmall" dataset shown in table6 which is

TABLE 5. Data specifications for real-life dataset and stream file.

	Dataset name: 100YWeatherSmall	
	Initial Dataset	**Extended Dataset**
Num. Documents	250,000	1,000,000(One million)
Avg. Document Size	1.7KB	1.7KB
Total Document Size	403.0MB	1.6GB
Num. Indexes	5	1
Total Index Size	11.3MB	9.5MB
Stream File Name(.txt)	Not req	objectId-weather
Stream File Size	Not req	26,368 KB

clearly presenting a complex schema. Other documents from same dataset contain different or same schema as presented by this table making this whole dataset an un-structured one.

C. JOIN MODULE

We present here, working of semi-stream join module for un-structured and structured data using different sized stream files and datasets stored on disk. For the implementation of join module, we first created synthetic datasets and copy real-life dataset from Atlas Cluster on local machine by establishing MongoDB connection using configurations shown in table 7. Following python string is used to establish the connection of stored datasets with the join module written in python:

local_host = ''mongodb://localhost:27017/''

After successful connection with dataset, join module reads random object_ids from corresponding stream file and stores them in stream_list. In order to develop a complete stream, five random fields(structured, un-structured) are also generated at run time and are appended into stream_list variable to be attached with each selected object_id. This stream_list will reside in memory for join process therefore size of stream_list is then added into overall memory usage. Stream documents from stream_list are then matched to the corresponding disk-based datasets using the following MongoDB query statement:

''collection.find('_id':ObjectId(stream_list[0]))''.

Once the match is found, stream documents are joined with the dataset documents and stored in a text file on disk. Each find() query is returning exactly one document matching with stream object id as there are no two documents matching with same object id. Same cycle repeats until all stream documents from stream_list find their match from dataset. Memory usage and CPU time are recorded for complete join operation for all testing arrangements using ''time'', ''psutil'' and ''os'' libraries alongside following python statements:

```
memory = psutil.Process(os.getpid())# for memory size
start_time = time.time()
# executing join module
execTime = time.time() − start_time
# print execution time in seconds
print(execTime)
# print memory size in bytes
print((memory.memory_info().rss)/(1024**2))
```

TABLE 6. One random disk-based document from collection ''data'' of ''100YWeatherSmall'' dataset.

No.	Fields	Type	Values	
1	id	ObjectId	5553a98ce4b02cf7150dee4e	
2	st	String	''x+44200-063500''	
3	ts	Date	1984-01-01 00:00:00.000	
4	position	Object	**Fields** type coordinates	**Values** ''Point'' Array 0 : -63.5 1 : 44.2
5	elevation	Int32	9999	
6	call Letters	String	''CG23''	
7	quality Control Process	String	''V020''	
8	data Source	String	''4''	
9	type	String	''FM-13''	
10	air Temperature	Object	**Fields** value: quality:	**Values** 999.9 ''9''
11	dew Point	Object	**Fields** value: quality:	**Values** 999.9 ''9''
12	pressure	Object	**Fields** value: quality:	**Values** 999.9 ''9''
13	wind	Object	**Fields** direction: type: Speed:	**Values** Object angle:220 quality:''1'' ''N'' Object rate:10.3 quality: ''1''
14	visibility	Object	**Fields** distance: variability:	**Values** Object value:20000 quality:''1'' Object value: ''N'' quality:''9''
15	sky Condition	object	**Fields** ceiling height: cavok:	**Values** Object value:9999 quality:''9'' determination: ''9'' ''N''
16	sections	Array	**Index** 0: 1: 2: 3:	**Values** ''AG1'' ''AY1'' ''GF1'' ''MW1''
17	precipitation estimated observation	Object	**Fields** discrepancy: estimated water depth:	**Values** ''1'' 0
18	present weather observation manual	Array	**Fields** 0:	**Values** Object condition:''02'' quality:''1''

We recorded of memory usage and execution time after five iterations for each arrangement in order to compute average

TABLE 6. *(Continued.)* **One random disk-based document from collection "data" of "100YWeatherSmall" dataset.**

No.	Fields	Type	Values	
19	past weather observation manual	Array	**Fields**	**Values**
			0:	Object
			atmospheric condition:	Object
				value:"0"
				quantity:"1"
			period:	Object
				value:6
				quantity:1
20	sky condition observation	Object	**Fields**	**Values**
			total coverage:	Object
				value:"00"
				opaque:"99"
				quality:"1"
			lowest cloud coverage:	Object
				value: "00"
				quality: "1"
			low cloud Genus:	Object
				value: "00"
				quality: "1"
			lowest cloud base height:	Object
				value: 99999
				quality: "9"
			mid cloud genus:	object
				value: "00"
				quality: "1"
			high cloud genus:	Object
				value: "00"
				quality: "1"

TABLE 7. **MongoDB connection configuration.**

	Real-life dataset	Synthetic dataset
Host name:	cluster0-shard-00-00-jxeqq.mongodb.net	localhost
Port:	27017	27017
Authentication:	Username/Password	None
Authenticate database:	Admin	None
Replica Set Name:	Cluster0-shard-0	-
Read Preference:	Primary Preferred	Primary
SSL:	System CA / Atlas Deployment	None

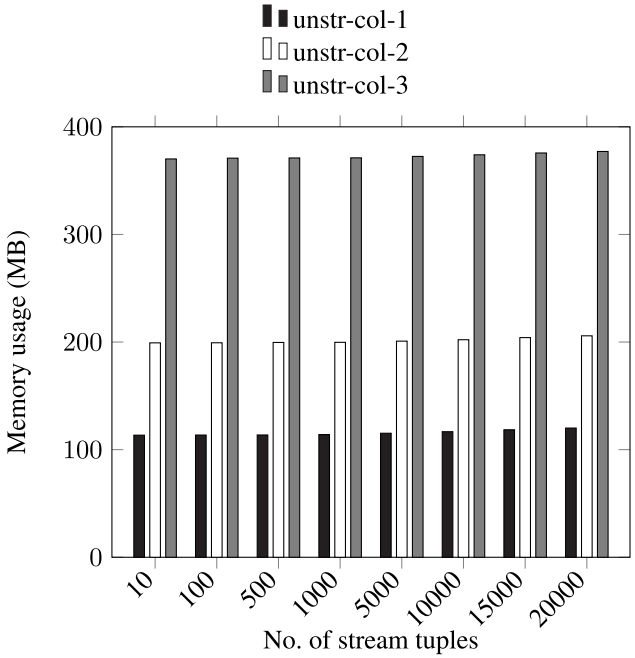

FIGURE 3. **Memory usage for stream processing using three different sized un-structured datasets.**

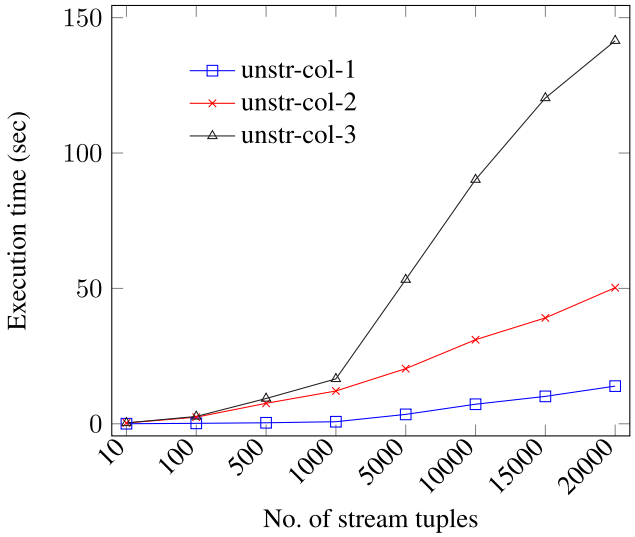

FIGURE 4. **Execution time for stream processing using using three different sized un-structured datasets.**

values as there can be any scenario among best, average or worst case match for each stream.

In this section, overall methodology is described in order to perform the proposed analysis. Implementation details, testing arrangements, data specifications and working of join module are explained in detail by this section.

IV. EXPERIMENTAL EVALUATION

In this section we present an experimental study comparing memory usage and execution time for semi-stream join based on two categories i.e, synthetic and real-life datasets. In order to obtain a range of results we use all stream sizes on each dataset mentioned in testing arrangement section. We present memory and execution time comparison for processing all defined stream sizes using synthetic un-structured and structured datasets respectively in sub-sections A-1 and A-2 individually, while connection between performance of join module using un-structured and structured datasets is being evaluated in sub-section A-3. Whereas sub-section B describes performance comparison for stream join processing using real-life un-structured dataset.

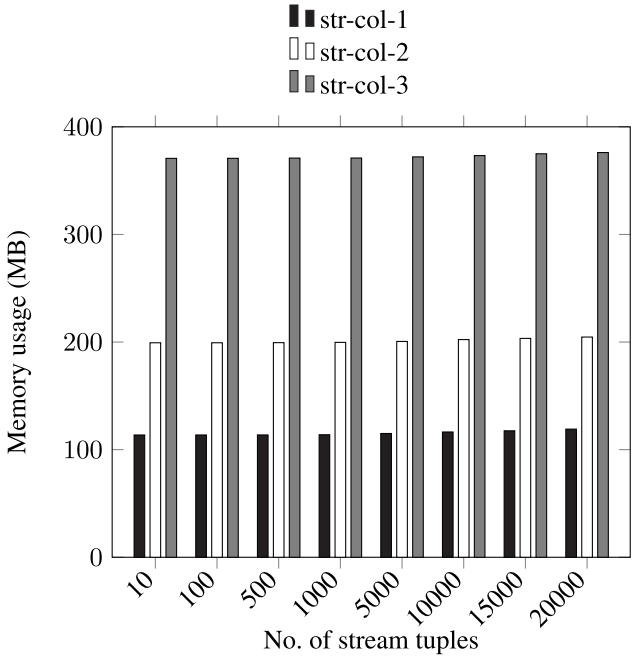

FIGURE 5. Memory usage for stream processing using three different sized structured datasets.

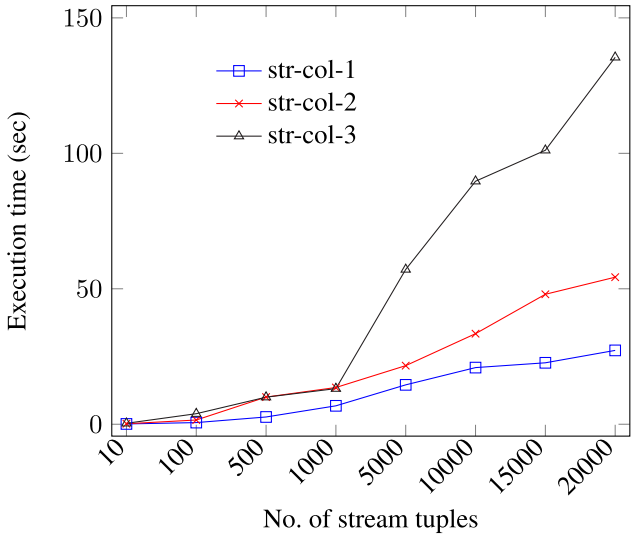

FIGURE 6. Execution time for stream processing using three different sized structured datasets.

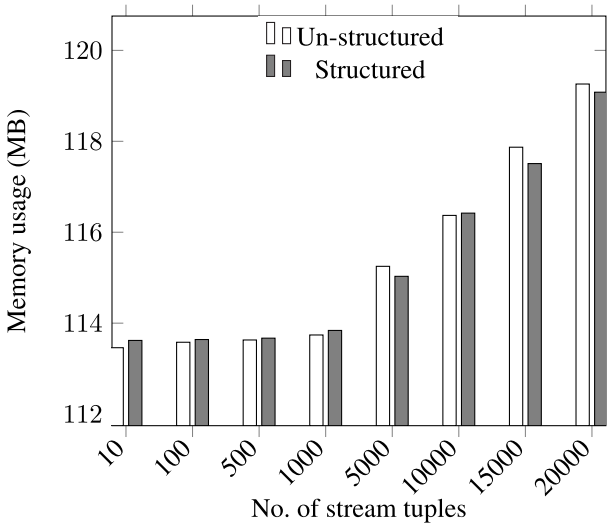

FIGURE 7. Memory usage comparison: ONE million no. of documents.

A. PERFORMANCE COMPARISON USING SYNTHETIC DATA

To study the behaviour of join module, synthetic datasets are being tested with all defined stream sizes. Un-structured and structured data sets are used in this evaluation in order to assess performance.

1) UN-STRUCTURED STREAM PROCESSING WITH DIFFERENT SIZED DISK-BASED COLLECTIONS

Figure 3 presents memory consumption comparison for stream join operation using un-structured collections. Disk-based collection name ''unstr-col-1'' takes less memory for stream join operation while more memory is required for other two collections as their disk sizes are increased (from 543.8MB to 1.1GB and 2.1GB). whereas almost constant rate is observed by this memory bar graph for increasing number of streams proving that increased rate of incoming streams does not require more memory for stream processing because these streams contain less attributes hence are of less size. We also compared execution time in seconds for stream join operation using all un-structured collections shown in figure 4. From figure 4 it can be noted that, join module process small number of streams in very less time for all sizes of disk-based collections. However, more execution time is required while increasing number of streams matching with increasing size of disk-based collections. This is not unexpected, since join of each stream can be any case among best/average/worst cases and where there is a huge size of disk-based collection to search in. It is concluded from the behaviour of join module with different sized collections and streams that almost same memory settings are required for one sized collection irrespective to the number if incoming streams, however more execution time is required to process more number of streams in case of larger datasets.

2) STRUCTURED STREAM PROCESSING WITH DIFFERENT SIZED DISK-BASED COLLECTIONS

We also evaluated performance of join module for different sized disk-based structured collections (str-col-1, str-col-2, str-col-3) on the basis of their memory consumption and execution time. Memory usage for stream processing using one, two and four million disk-based collections is compared and presented in figure 5, which shows consistent rate for increasing number of streams proving that increased rate of incoming streams does not require more memory, however more memory is required in case of each larger dataset. Figure 6 presents comparison of execution time for stream

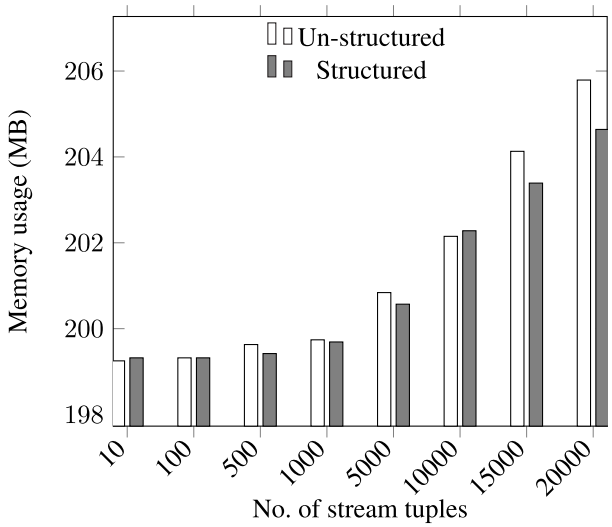

FIGURE 8. Memory usage comparison: TWO million no. of documents.

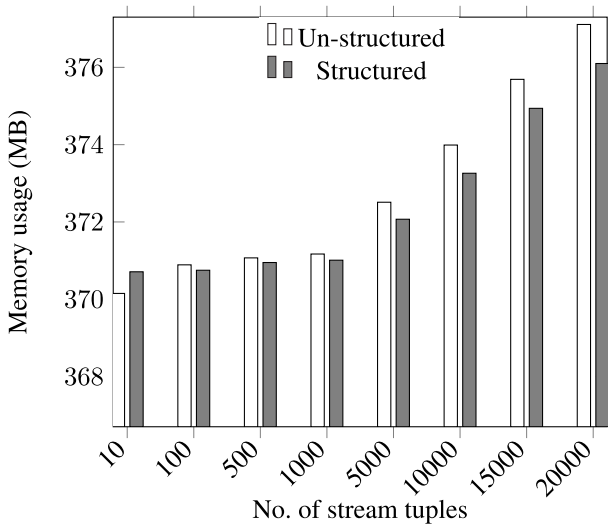

FIGURE 9. Memory usage comparison: FOUR million no. of documents.

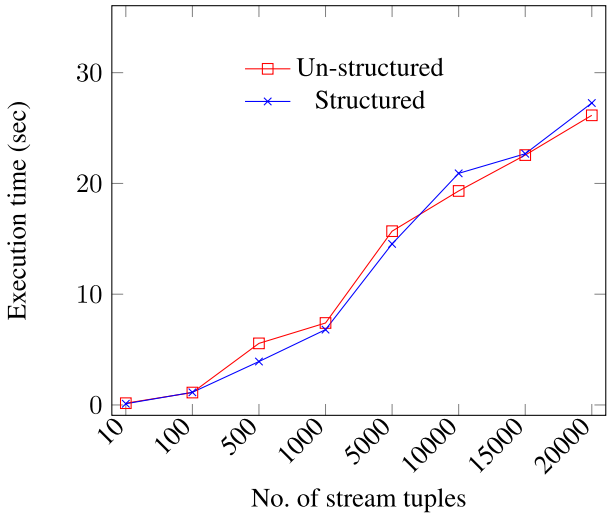

FIGURE 10. Execution time comparison: ONE million no. of documents.

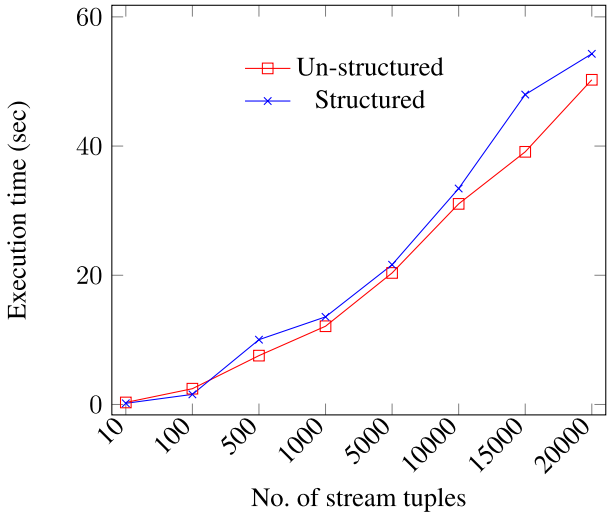

FIGURE 11. Execution time comparison: TWO million no. of documents.

join operation with different sized disk-based structured collections and using different sized streams. The results in figure 6 show that as soon as the number of streams increases join module takes more time for each dataset. Moreover, more execution time is required in order to match streams with comparatively larger datasets as each stream needs to find its match starts from the beginning of dataset till it is found i.e, last page of dataset for worst case scenario.

3) UN-STRUCTURED VS STRUCTURED STREAM PROCESSING

In order to examine the connection between un-structured and structured stream processing, we prepared memory usage bar graphs shown in figures 7, 8 and 9 and execution time line graphs are shown in figures 10, 11 and 12. Memory graphs depict that almost same memory is required to join a certain number of streams using structured or un-structured disk-based collections for sizes one, two and four million documents. However, it can be noticed that at 5000 or more stream tuples slightly more memory is required to process un-structured streams as average size of each un-structured document is more than that of structured document (see tables 2 and 3). Approximately similar execution time is observed for each testing arrangement using both un-structured and structured datasets showing that stream processing takes same execution time irrespective to the nature of stream (un-structured/structured), results in execution time graphs are in support of our findings. However, slight differences among execution time with the use of un-structured and structured datasets are due to unrecorded best/average/worst case scenarios.

B. PERFORMANCE COMPARISON USING REAL-LIFE DATA

In these experiments we measured memory usage and execution time of our join module using real-life data and then present a comparison with the results of synthetic data in

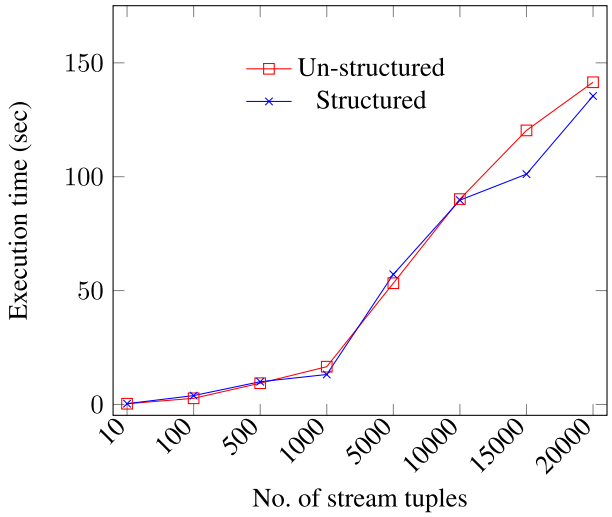

FIGURE 12. Execution time comparison: FOUR million no. of documents.

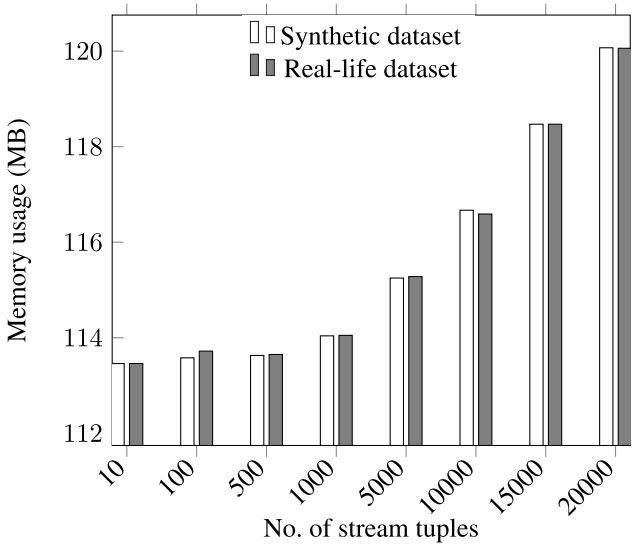

FIGURE 13. Memory usage comparison: (un-structured) synthetic and real-life datasets with one million no. of documents each.

order to validate our findings. Memory usage comparison shown in figure 13 clearly depicts that same memory settings are required for both categories of datasets because both datasets contain un-structured documents. Execution time comparison of real-life dataset with synthetic dataset is shown in figure 14 using one million number of documents in each case. We note that, for higher number of stream tuples, the execution time of join module using real-life dataset is much higher than using synthetic dataset because of bigger size of real data documents i.e, 1.6GB where as synthetic data size of one million documents is 543.8MB (see tables 3 and 5 [extended dataset]). In support of it, we have found that certain sized pages are brought into page cache by mongodb.[2] Hence less number of documents can reside in one page for

[2]https://docs.mongodb.com/manual/administration/analyzing-mongodb-performance

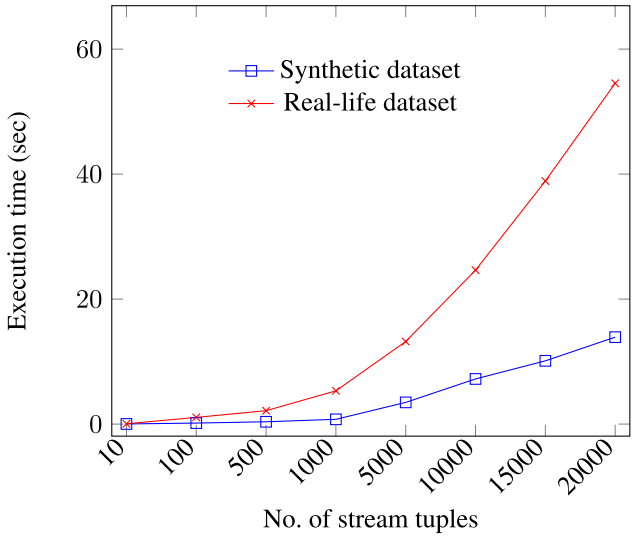

FIGURE 14. Execution time comparison: (un-structured) synthetic and real-life datasets with one million no. of documents each.

big sized documents therefore more disk I/O operations are required for worst case which causes more execution time for loading/unloading disk pages increasing overall execution time for our join module.

When considering the performance of join module using defined testing arrangements, we can predict the behaviour of our implementation for increasing sizes of streams and datasets. More memory settings are required for increasing size of datasets (un-structured/structured) while more execution time is required for growing stream tuples.

V. CONCLUSION

In this paper, we have identified challenges in joining NoSQL streams after analyzing semi-stream join processing through un-structured and structured streams of data coming with different I/O rates i.e, one stream from memory and other is from disk. We implemented a join module and multiple datasets in order to perform this analysis based on two parameters, CPU and memory utilization. In this implementation, We first generated six different sized un-structured and structured disk-based collections(tables) and one real-life un-structured dataset and then created their corresponding stream files. Streams from stream files reside in memory and are fetched one by one to find their match from disk-based big sized dataset based on primary key and then recorded memory and execution time for this join operation based on testing criteria. Our experimental results showed that under the given configuration, a fixed sized memory and constant CPU time is required for stream join processing for a given stream size, irrespective of the nature of data stream, which however increases for growing stream size. Stream processing with larger datasets need more disk I/O therefore increasing execution time and memory used. This paper plays a role as a guideline for data streams join operations using NoSQL database MongoDB and help developers to choose the appropriate sized memory for fast incoming

un-structured/structured streams by reducing disk I/O cost for successful implementation of real-time data warehousing.

REFERENCES

[1] R. Yangui, A. Nabli, and F. Gargouri, "Etl based framework for NoSQL warehousing," in *Proc. Eur., Medit., Middle Eastern Conf. Inf. Syst.* Cham, Switzerland: Springer, 2017, pp. 40–53.

[2] N. Sharma, A. Iyer, R. Bhattacharya, N. Modi, and W. Crivelini, *Getting Started With Data Warehousing*, 1st ed. Victoria, BC, Canada: IBM Canada, 2012.

[3] P. Ponniah, *Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals*. Hoboken, NJ, USA: Wiley, 2004.

[4] K. B. S. Kumar, Srividya, and S. Mohanavalli, "A performance comparison of document oriented NoSQL databases," in *Proc. Int. Conf. Comput., Commun. Signal Process. (ICCCSP)*, Jan. 2017, pp. 1–6.

[5] Z. Bicevska, A. Neimanis, and I. Oditis, "NoSQL-based data warehouse solutions: Sense, benefits and prerequisites," *Baltic J. Mod. Comput.*, vol. 4, no. 3, p. 597, 2016.

[6] Z. Bicevska and I. Oditis, "Towards NoSQL-based data warehouse solutions," *Procedia Comput. Sci.*, vol. 104, pp. 104–111, 2017.

[7] Z. Parker, S. Poe, and S. V. Vrbsky, "Comparing NoSQL MongoDB to an SQL db," in *Proc. 51st ACM Southeast Conf.* Savannah, Georgia: Univ. Savannah, 2013, p. 5.

[8] H. Mallek, F. Ghozzi, O. Teste, and F. Gargouri, "BigdimETL with NoSQL database," *Procedia Comput. Sci.*, vol. 126, pp. 798–807, 2018.

[9] A. Flores, S. Ramírez, R. Toasa, J. Vargas, R. Urvina-Barrionuevo, and J. M. Lavin, "Performance evaluation of NoSQL and SQL queries in response time for the e-government," in *Proc. Int. Conf. eDemocracy eGovernment (ICEDEG)*, Apr. 2018, pp. 257–262.

[10] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *Proc. IEEE Pacific Rim Conf. Commun., Comput. signal Process. (PACRIM)*, Aug. 2013, pp. 15–19.

[11] D.-H. Shih, F.-C. Huang, W.-H. Lai, and M.-H. Shih, "Privacy preserving and performance analysis on not only SQL database aggregation in bigdata era," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Apr. 2017, pp. 56–60.

[12] D. Sahiet and P. D. Asanka, "ETL framework design for NoSQL databases in dataware housing," *Int. J. Res. Comput. Appl. Robot.*, vol. 3, pp. 67–75, Nov. 2015.

[13] W. Wingerath, F. Gessert, S. Friedrich, and N. Ritter, "Real-time stream processing for big data," *Inf. Technol.*, vol. 58, no. 4, pp. 186–194, 2016.

[14] P. Córdova, "Analysis of real time stream processing systems considering latency," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2015.

[15] R. Tönjes, P. Barnaghi, M. Ali, A. Mileo, M. Hauswirth, F. Ganz, S. Ganea, B. Kjærgaard, D. Kuemper, S. Nechifor, and D. Puiu, "Real time IoT stream processing and large-scale data analytics for smart city applications," in *Proc. Eur. Conf. Netw. Commun.*, 2014, pp. 1–5.

[16] 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.

[17] M. A. Naeem, G. Dobbie, and G. Weber, "HYBRIDJOIN for near-real-time data warehousing," *Int. J. Data Warehousing Mining*, vol. 7, no. 4, pp. 21–42, 2011.

[18] 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.

[19] 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.

[20] E. Mehmood and M. A. Naeem, "Optimization of cache-based semi-stream joins," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Chengdu, China, Apr. 2017, pp. 76–81.

[21] M. A. Naeem, G. Weber, and C. Lutteroth, "A memory-optimal many-to-many semi-stream join," *Distrib. Parallel Databases*, pp. 1–27, Aug. 2018.

[22] S. M. Freire, D. Teodoro, F. Wei-Kleiner, E. Sundvall, D. Karlsson, and P. Lambrix, "Comparing the performance of NoSQL approaches for managing archetype-based electronic health record data," *PLoS One*, vol. 11, no. 3, 2016, Art. no. e0150069.

[23] G. Delussu, L. Lianas, F. Frexia, and G. Zanetti, "A scalable data access layer to manage structured heterogeneous biomedical data," *PLoS One*, vol. 11, no. 12, 2016, Art. no. e0168004.

[24] Y.-H. Jeon, K.-H. Lee, and H.-J. Kim, "Distributed join processing between streaming and stored big data under the micro-batch model," *IEEE Access*, vol. 7, pp. 34583–34598, 2019.

ERUM MEHMOOD was born in Pakistan. She received the M.Phil. degree in computer science from NCBAE, Lahore, Pakistan, in 2017. She is currently pursuing the Ph.D. degree with the University of Management and Technology, Lahore, Pakistan under the supervision of Dr. T. Anees. Her M.Phil. dissertation was on the area of stream processing for real-time data warehousing.

She is currently 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.

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 was on the area of service-oriented architecture and web services availability domain. She was a Project Assistant with the Vienna University of Technology for four years. She is currently the Director of the Software Engineering Program and an Assistant Professor with the Software Engineering Department, University of Management and Technology, Lahore. Her research interests include service-oriented architecture, web services, software availability, software safety, software engineering, software fault tolerance, and real-time data warehousing.