International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-2, December 2019

Innovative Web Service Grammar for Streaming Computing in Real-Time Technology

Boumedyen Shannaq

Abstract*: The purpose of this work is to develop a UJSON grammar for the distributed system in Telecommunication and sensor network technology. Execute continuous requests on JSON streaming data based on advanced technologies for parallel streaming computing, suitable for solving analytic problems and calculation of metrics in real-time. The developed UJSON grammars in this research work designed to filtering event flow, building an event flow as a query result, grouping and aggregation of events, and creating window semantics. For testing the proposed work, several queries were selected that implement aggregation with different types of semantic windows (Steps, Slides). Testing was done locally and on commercial hypermarket clusters. It was used 4 types of configurations 2, 4, 8, and 16 computing nodes. Based on the obtained results, scalability is noticeable with an increase in the number of nodes. The updated functions of the proposed UJSON could improve the construction of parallel flow systems and data processing. The developed approach based on modern and advanced parallel flow technologies for output calculations considering the pros and cons of various approaches found in the current era.

Keywords* : : JSON, Real-Time, Streaming Computing, Web Service Technology.

I. INTRODUCTION

The development of Internet technologies has given a new impetus in the path of parallel programming which recently has acquired new consumer qualities. This led not only to the obvious progress of technologies and programming languages. This actually created an inverse effect on the understanding of the parallel process. [1] Parallel programming technologies have changed dramatically, the initial use of computer devices as calculators smoothly switched to using them as information processors. The architectural solutions gave way to semantics and flexible distribution of software functionality among “hardware executors”. The first purpose of computers: complex mathematical calculations, industrial applications and everything that did not concern everyday life, mobility, and the Internet. Naturally, when the tasks of parallel programming are so "limited", it is difficult to expect interesting achievements.

Revised Manuscript Received on December 30, 2019.
***** Correspondence Author
Dr.Boumedyen Shannaq*, MIS, Business College, University of Burimi, Al Burimi, Oman. Email: boumedyen@uob.edu.om

When computers became mass-produced, the Internet and mobile devices appeared, the requirements for parallelism changed dramatically, and the developers had to radically change the style and speed of work. The first signs were the idea of messaging between processes for clusters where each computing node is running its own operating system parallel to a program, which is a set of processes, are required mechanisms for transmitting data over a network. In each of the above cases, the developer cannot do without support from the operating system, which should provide the necessary API for working with streams, synchronization primitives, mechanisms for exchanging data between processes that run as on one computing node, and on different. The expected effectiveness of special technologies developments offering support either in the form of function libraries or at the level of the compiler. The most common way to create parallel programs for shared memory systems is through the use of threads [2]. At the same time, functional concurrency is easily provided by writing different stream functions, and data parallelism are realized. The programmer can work with streams, both using the API of the operating system, and creating own stream library. The latter approach in some cases can provide more program performance due to lower overhead, but much more time-consuming. Based on the review, we can distinguish a general scheme in advanced control systems databases. Existing Continuous Query Systems over streams of data in JSON format does not satisfy all requirements since they are not pure stream processing systems [3]. For example, there is no correct format with the specification, the JSON format does not specify a format for exchanging dates, so there are so many different ways to do this. They are either based on any database management system and require mandatory storage of events in it, which can significantly slow down processing, or do not provide a streaming model of continuous requests, and only periodically recalculate its result, which contradicts the most urgent processing of incoming events [4]. Thus, it seems advisable to develop a new system that could perform continuous queries on JSON streaming data based on modern technologies for streaming computing. Unlike XML, for which it is already developed enough a large number of languages for streaming processing, for example XSeq [5] and XQuery [6].JSON does not have popular languages for specifying continuous requests. In essence existing systems use either self-developed languages, or borrowed from other systems. For example, the language AmosQL [7] object is used DBMS, the request is specified by sending a JSON request to the server executing the requests. Based on this situation, this work decided to develop a suitable language using JSON structure for job continuous query.

II. BACKGROUND AND LITERATURE REVIEW

In the Telecommunication networks of Internet access providers, networks of mobile operators require real-time streaming processing, such as monitoring the status of the network and equipment, identifying and defeating malicious activity on the part of users, as well as the collection of statistics on the use of the network and its load [8]. In Sensor Networks, data comes from a variety of sensors or devices (Splitter), which periodically generate some events. For example, this may be a GPS receiver mounted on a car sending its coordinates every minute, or RFID scanner [9] tags at the exit from the library systems, which keeps track of all the books taken out of its locations. Collecting information from multiple sensors and processing it in automatic mode, could build systems that monitor the correct functioning and managing complex processes of the real world, virtually and without human intervention. In today's high-load applications and web services [10], tasks flow loud do not lose their relevance. In particular, problem-solving is required. calculation of financial indicators in real-time, such as amount, average price transactions for the last day; identifying popular content - identifying the most frequently mentioned recent news; service status monitoring - tracking the number and types of requests, the answer to which requires too much great time. The success of a business depends on the quality of solving such problems. Processes and entire companies in which they arise. Therefore there is a whole area of systems designed to simplify solving analytic calculation problems and real-time metrics [11]. complex systems for processing large amounts of data in highly loaded web applications has become a cluster of homogeneous machines that do not share resources (shared-nothing), interconnected by a regular network, for example, in Ethernet there is no special support from the equipment side for such an architecture for distributed computing, and therefore all the care of organizing and parallelization management falls on the software part. It caused rapid development of the field of distributed computing, as a result of which a large stack of open source software was developed, making it easy to develop programs requiring scalable and fault-tolerant. Pioneers in this area can be considered Google, inside of which technologies that are widely used today have been developed MapReduce [12][13] and the 'Google File system ' GFS [14], and many others. Big data also appears in streaming tasks, for example, often the arrival of events in some streams can reach tens, and sometimes a hundred thousand events per second. And for such tasks, requirements for parallel appeared stability, scalability, and fault tolerance of their decisive systems. And consequently required the use of technology for distributed computing. In connection with this in the last few years began to appear solutions, specialized and for organizing distributed streaming computing such as S4 [15] and Storm [16]. But they are still quite young and on the basis of them, a system has not yet been developed, Designed to perform continuous queries on streaming data. The Storm framework can be considered one of the best Big Data solutions when it comes to open-source platforms.[17] Unlike Hadoop and Spark, focused on batch processing of large data sets, the Storm system is designed for distributed processing in real-time and does not depend on the programming language. Workflows in Storm are called 'topologies' these topologies are organized by the principle of a directed acyclic graph (DAG) and are executed until the user shuts down or a fatal error occurs. Storm supports the creation of topologies that transform incomplete data streams. These conversions, unlike Hadoop jobs, never stop, but continue to process data as it arrives. Native Storm cannot be used to develop Big Data applications on top of typical Hadoop clusters. To coordinate tasks between nodes in a cluster, Apache ZooKeeper is used with its master (minion) worker. However, Yahoo! and Hortonworks are working on creating libraries to run Storm on top of Hadoop 2.x YARN clusters. Also, note that this framework is able to read and write files from/to HDFS.JSON or JavaScript Object Notation is a format that implements an unstructured textual representation of structured data, based on the principle of key-value pairs and ordered lists. Although JSON began its distribution with JavaScript, it is supported in most languages, either natively or through special libraries. Typically, JSON is used to exchange information between web clients and a web server. To understand the usefulness and importance of JSON, we need to understand a little bit about the history of interactivity on the Internet. In the early 2000s, the interactivity of websites began to change. At that time, the browser served only to display information, and all the work on preparing the content for display was performed by the webserver. When the user clicked a button in the browser, the request was sent to the server where the HTML page was collected and sent, ready for display. Such a mechanism was slow and inefficient. This required the browser to redraw everything on the page, even if a small portion of the data changed initially, the data was transmitted in XML format, but it was difficult to use in JavaScript. JavaScript already had objects that were used to represent data in the language, so [18] took the syntax of JS objects and used it as a specification for a new data exchange format called JSON. This format was much easier to read and parse in a JavaScript browser. Soon, developers started using JSON instead of XML [19] as shown in figure 1. Nowadays, fast JSON data exchange is the de facto standard for transferring data between server and client, mobile applications and even internal system services. The main alternative to JSON was and is XML. However, XML is becoming less common in new systems.

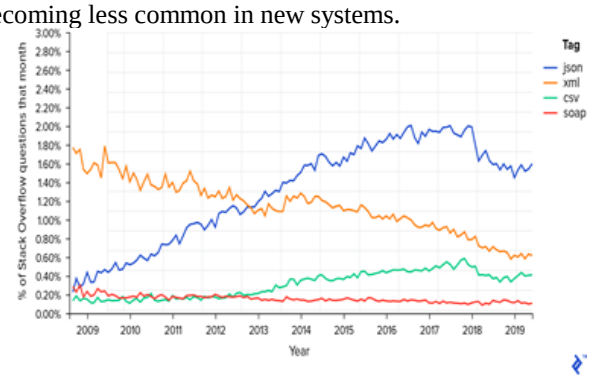

Fig.2. JSON,XML,CSV,SOAP statists use

[20] In addition to code redundancy, in fact, data writing took up twice as much space, XML still introduces some ambiguity in the analysis of data structure.[21] Converting XML to a JavaScript object can take from tens to hundreds of lines of code and requires fine-tuning for each parsed object. The conversion of JSON into a JavaScript object is performed in one line and does not require any prior knowledge about the analyzed object.

Although JSON is a relatively compressed and flexible data format that is easy to work within many programming languages, it has some disadvantages. Here are some limitations: There is no structure. On the one hand, this means that you have complete flexibility to present data in any way. On the other hand, you can easily store unstructured data. Only one type of numbers. IEEE-754 floating-point and double precision format supported. This is quite a lot, but you can't use the variety of number types that other languages have. No date type. Developers should use string representations of dates, which can cause formatting mismatch. Or use as the date the number of milliseconds that have passed since the beginning of the UNIX era. No comments - you cannot annotate fields that require this directly in the code. Verbosity - Although JSON is less verbose than XML, it is not the most concise data exchange format. For high-performance or specialized services, you will want to use more efficient formats. Despite the fact that the work is focused on poorly structured data in JSON format, many existing systems are designed to work with streams of events with a special and sometimes strictly defined structure, which, however, does not make them interesting to consider in the current technology demands (big data, homogeneity requests).Streaming data processing systems are built for different purposes. Some include only filtering suitable events at the maximum possible speed, while others require a complex recount of various hundred a logic that sometimes requires access to external data sources because the architecture is fundamentally different. In particular, a prominent representative of such systems is TelegraphCQ [22], this is a database group at the University of Berkley based on PostgreSQL.Another example is a JSQ system for performing continuous queries.over data streams in JSON format, based on the AmosQL DBMS [7]. The fact is that incoming events must be stored in the DBMS before processing. Some used statements in DBMS, such as order by, join- cannot be used unchanged to perform continuous requests, their result is determined only in those cases when the complete set is known input data, which is impossible with the constant arrival of new events. In XML Streaming Systems Large sub-area in stream processing and continuous. Owl is event processing in XML format, which allows storing weakly structured data. The popularity of this area is associated with convenience and the vastness of XML as a format for the interoperability of various applications between themselves Systems to perform continuous queries over XML began to develop in the late 1990s years, and are developing to this day. The general principle for performing continuous requests is no different from that in RDBMS-based systems: first, you need to build an implementation plan request, if possible to optimize it and then transfer it to the executor, to who will receive events and calculate the result. The differences are the fact that now incoming events are not records with fixed field's type, and objects with a potentially complex and varying hierarchical structure to work with many continuous query systems have been built to solve specific tasks, and therefore their functionality may be limited by needs of the task. Nevertheless, they are also of interest.

III. PROPOSED SOLUTION

If To develop a system for performing continuous queries on semi-structured streaming data in JSON format based on modern technologies for parallelization of stream computing. This task can be decomposed into subtasks as described in fig 2.

Fig. 2. Steps of parallelization of stream computing

In general, User interaction with the system should be done by writing queries in a special language. Therefore, it is necessary to choose such a language with continuous requests, which will be expressive enough to describe the entire set of valid query parameters that can be executed by the system. It is possible either to choose such a language from existing ones or to develop a new language. Unlike XML, for which it is already developed enough in a large number of languages for streaming processing, for example, XSeq and XQuery3.0, JSON does not have popular languages for specifying continuous requests. In essence, existing systems use either self-developed languages or borrowed from other systems. For example, the language AmosQL [7] object is used Amos II DBMS, the request is specified by sending a JSON request to the server for executing the requests. Based on these circumstances, it was decided to develop a suitable new language for the continuous query.

A. Description of the developed language

Since JSON is selected as the format of incoming events, the designed language must be able to parse the data in this format in order to extract the required language.

Fig. 3, describes the three types of expressions

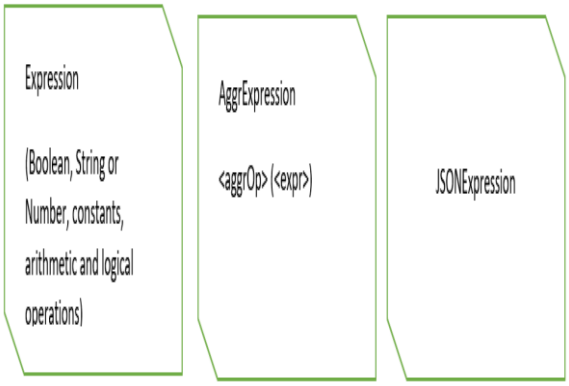

Fig. 3. UJSON sample types of expressions

Table-1: sample of the non-functional requirements

1	The system should support distributed execution on clusters of their heterogeneous machines
2	The system has a purely streaming processing model that does not require costly operations with storing events to disk
3	The system should take into account the value of a field that is not in the received event
4	The system must correctly handle Incorrect request events and not terminate the request with an error
5	Events should not be saved
6	Events must be handled by continuous requests, and only they have to decide whether to save the event or not
7	The system should Guaranteed reliability of data processing
8	The system should enable the Automatic scaling and rebalancing
9	The system should handle Immediate event processing and output

B. The choice of the framework

Any streaming processing task, including performing the continuous request, can be represented as processing events from a stream. [23][24] demonstrate different implementation, In turn, each event can be broken down into a sequence of steps. Every step can be represented as a function written in some programming language. At the input, this function receives an event and returns a set of new events. (Possibly empty, depends on the task), which must be passed to the function trace step to continue processing. In addition to the functions themselves, they are responsible for following the steps, for a complete description of the task, storing the sequence of steps that must be performed in order to boots of each event. In this model, complete information about the task over streaming data is called a topology (includes the functions of steps and configuration of their sequence). S4 frameworks have been selected in this work, S4 accepts input. Topology and run it on the cluster. Moreover, for each function of the steps, multiple copies can be created, distributed across several nodes to achieve parallel execution of the giving task and, accordingly, leads to greater productivity. With this approach, each cluster node will perform multiple instances of various functions. Messaging between features steps occurs through the message queues provided by the framework without human intervention. In addition, support frameworks aware of cluster reconfiguration and load balancing. To implement the developed system, this work chooses STORM framework, which could provide facilities for ensuring the reliability of execution and recovery from failures.

C. The Choice of algorithms for implementation in the system

After determining the technological base on which the system will be developed. This work develops algorithms for performing continuous queries based on the selected Storm framework. Depending on the type of continuous request, the topology for its execution may It can have a different look. But first, we show how to describe the topology for the Storm framework and then we describe the updated topology. Fig. 4, describe the Storm topology for continuous request in a simple case.

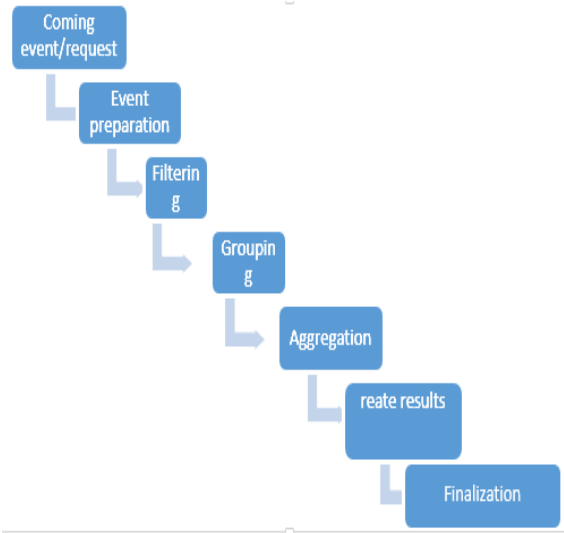

Fig.4. Storm topology for continuous request in a simple case

The process of processing each event when executing continuous requests could be achieved by break down the described steps in fig. 6, the Event preparation adds information necessary to support window semantics; the Filtering calculates the expression by the received event and passes it for further processing or discards; Grouping calculates the UJSON expression corresponding to the group key, to which this event belongs. Passes this key for further work together with the message; Aggregation calculates the result of one aggregation operation; Creating a result - accepts input or the results of operations or incoming events, and on them builds an event in a given output format; Finalization delivers of an updated result of a continuous request to the caller, and also considers official statistics, such as processing time events, bandwidth topology. In the simple case of the storm topology, when it is not required to perform aggregation operations, performing a particular query can be reduced only to filtering the flow of incoming events and their transformation. The topology for this case is shown in Fig. 4. Note that performing a grouping operation in the simple case does not make sense, because in any case, all events fall into the output stream. If the request contains aggregation operations, the topology will take the structure shown in fig.6.Basically, there are two parameters that can characterize a window semantics in the continuous query. The first is a way to set the window size. Usually, it is measured either by the number of events received (the window includes last N events) or by their age (events that came over the last N seconds/minutes/ ...). The second is the "behavior" of the window. There are two main ways: either window, in fact, are not used, and the whole stream is one big window or the window “slides” upstream to the right with the arrival of new events, and the oldest are excluded from the window ("sliding window").

Or the window "Steps" along the stream - after the window is fully processed, all events discarded from it and the window is filled with new events, thus, each event will be counted only once ("walking" window). For visibility, the principle of operation of the two types of windows is shown in fig.5 and figure 6.

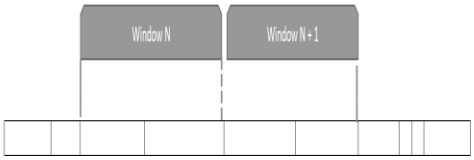

Fig.5. Steps Window

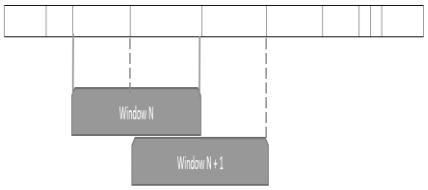

Fig. 6. Slides Window

Note that the complexity of performing aggregation operations in a stream is highly dependent. From the selected parameters of window semantics, namely the type of window -In case of non-use of window semantics, the task does not provide special difficulties, since aggregation operations only update their result based on the obtained values . In the case of steps windows, the only complication is achieved by that aggregation operations need to reset their value upon reaching window borders; The task is to synchronize them with each other so that everyone drops value at the same time. But every subroutine for performing aggregation knows parameters of window semantics, namely window length and by incoming messages can understand when the border of the current window comes. Therefore, the solution to this problem also is not particularly difficult. The most non-trivial case occurs when using sliding windows type - in this case, in addition to accounting for new messages, it is required to remove from consideration, old messages when their time comes to leave the window. The subroutine for the execution of the aggregation operator contains priority the sequence of values for the aggregation entering it, indicating the serial number and time. If the window size is determined by the number of events, then when a new one is added, the queue is viewed and obsolete values are deleted. If the window size is parameterized by time, then an additional thread of execution that removes values from aggregation results simultaneously with their obsolescence.

D. Description of the practical part

The practical part of the work was implemented in the Scala programming language tool. This choice is due to the fact that Storm Distributed Computing Framework is based on the developed system, implemented in the Java language. Scala, in turn, is implemented on top of Java virtual machine, which provides two-way compatibility with Java programs and libraries. The Scala language supports both object-oriented and functional programming paradigm, and also provides a number of high-level capabilities, which greatly increases efficiency, reliability and expressiveness of the developed code, which is quite important property of a programming language. The General scheme of the proposed work is described in fig. 7.

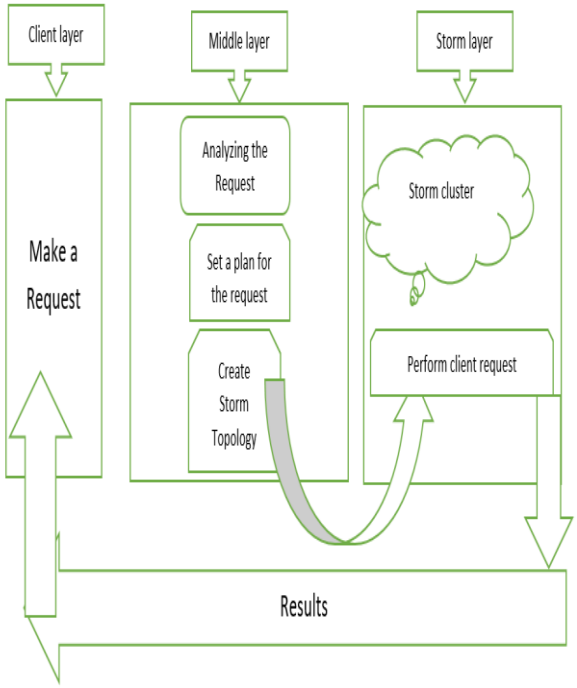

Fig 7. General scheme

Upon receipt of a continuous request to the system, its processing consists of two phases: building a query execution plan and building a Storm topology for this request. During each of these phases, a search is made for errors in the request and, when their detection, an error message is issued and the work with the request is terminated. Analysis of a continuous request and the construction of a plan for its implementation is made in one run. A continuous request execution plan contains a description of the form that of outgoing request events, expressions for filtering and grouping the stream incoming events, window semantics parameters, as well as parameters of each of aggregation operations, such as the expression by which the aggregation is performed, and their types. After parsing the request, the constructed plan for its execution is transmitted to the input of the topology of the Storm topology. Events from incoming data streams can be sent to the system as HTTP POST request for a special address. Continuous requests are also specified using POST requests. Events corresponding to the results of their execution are bounced back to the user using the same communication protocol over which they were asked.

E. Testing the developed system

Continuous query execution systems should be evaluated based on their productivity performance. The performance of streaming processing systems can be evaluated using two important parameters: bandwidth (in events per second) and for delay/interval between the time the event arrives and the time it's taken into account as a result of the continuous request. Depending on the type and complexity of the request, these characteristics can greatly vary. Therefore, for testing, several queries were selected that implement aggregation with different types of windows.

As test data was used and the generated event stream was used as test data describing transactions in some stores. Testing was done locally and on commercial hypermarket clusters. It was used 4 types of configurations 2,4,8, and 16 computing nodes. Testing was done locally and on commercial hypermarket clusters. It was used The results in Bandwidth (event per second) test results presented in Bandwidth Graph. fig. 8, presents a graph of the throughput abilities on the number of nodes used.

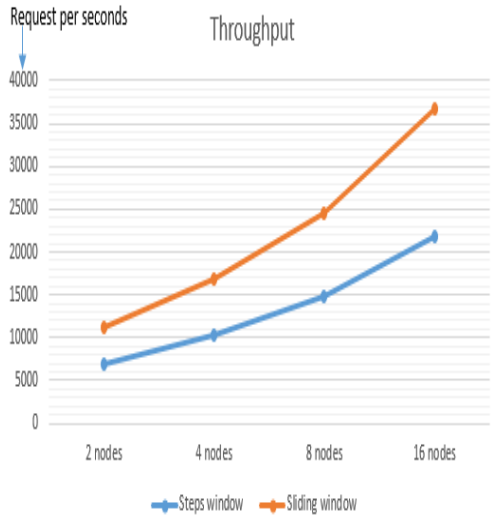

Fig.8. Throughput

Based on the obtained results, scalability is noticeable with an increase in the number of nodes. The average time for complete event processing (in seconds) is presented in fig.9.

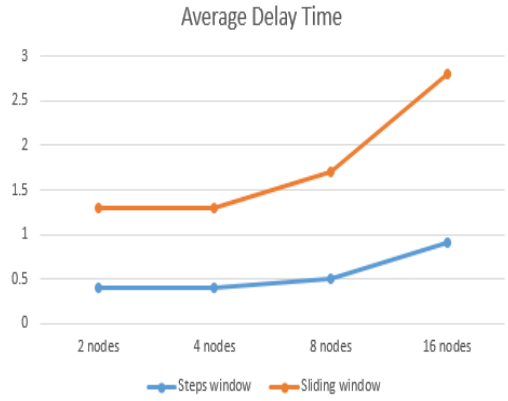

Fig. 9. Average Delay Time

IV. DISCUSSION

Any system in which the generation of the output signal plays a significant role. This is usually due to the fact that the input signal corresponds to some changes in the physical process, and the output signal must be associated with the same changes. The time delay from receiving the input signal to issuing the output signal should be small in order to provide an acceptable reaction time. Reaction time is a system characteristic: when controlling a rocket, a reaction takes several milliseconds, while dispatch control of the movement of ships requires a reaction time, measured in days. Systems are usually considered real-time systems if their reaction times are on the order of milliseconds; Interactive systems are considered to have reaction times of the order of several seconds, and in batch processing systems, reaction times are measured in hours or days. Examples of real-time systems are physical process control systems using computers, vending machine systems, automated control systems, and automated test complexes. To work with such data volumes, new technologies are needed. It is obvious. If a company needs to process "big data", then, they can simply increase the available capacity (vertical approach), that is, replace existing equipment or upgrade it. However, this is a dead-end approach due to the rapid growth in data volume. On the other hand, you can buy more servers and computers and distribute the load between them (horizontal approach). So you can create a highly reliable distributed network with a total capacity superior to supercomputers. Although this approach is not universal. Not everyone who works with "big data" has the opportunity to create a fleet of tens, hundreds or even more servers. What to do? It is worth looking at cloud technologies that allow you to simultaneously apply both approaches. In fact, at present, it is cloud solutions that most fully meet the requirements of processing technologies of "big data". Clouds are easy to scale, they can manage huge data storage systems, and they can redistribute the load geographically and transfer data at the highest available speeds now. They can create virtual supercomputers, which, if necessary, increase their power without interruption in work. Companies can make their own private clouds, they can buy public resources or build hybrids. UJSON needs to be done not always, but either in cases where it is required that not only the browser work with the API, but also mobile clients or some other ones (which do not need ready-made data in HTML, but need JSON for parsing), or when it's better than the customer is fat. Even UJSON, obviously, makes debugging a little easier. In general, ideally, the back-end of the site should be able to return both HTML for the browser (which JS does not need to parse => performance on the client), and UJSON for other purposes, and maybe something else. UJSON is a convenient data serialization format it's built on the basis of JSON. The ability to serialize/deserialize in JSON is literally in all programming languages, which makes this format quite universal. If you do not use it, you will have to implement something else. It can be other serialization formats, for example,' message pack', or if you want, pack the data yourself as you want, depending on the task. But in this case, you have to implement serialization yourself, and this is the time. And time is money. That's why it should pay off (for example, to increase the bandwidth using your binary serialization format developed exclusively for the task.

V. CONCLUSION

As part of this work, all the tasks have been performed and their results were obtained. A proposed language UJSON of continuous queries has been developed for streaming data in the format of JSON, supporting the Filtering event flow functionality, Building an event flow as a query result, Grouping and aggregation of events and Window semantics have been demonstrated and tested. A system has been developed for performing continuous requests specified at a time query language based on modern technologies. The obtained results show a visible improvement over using the UJSON in real-time technology.

The value of the outcomes of this work is for solving urgent tasks of analytics and calculating real metrics time, developed on the basis of modern technologies for parallel stream computing.

REFERENCES

1. Shannaq B., Adebiaye R.”Schemes for Distributed Computing Environment Based on Cloud Computing Technology for Ministry of Regional Municipalities and Water Resources (MRMWR) Oman”,http://www.ijeijournal.com/papers/Vol.6-Iss.1/A06010106.pdf,2016.
2. William S.”Operating Systems: Internals and Design Principles”, Global Edition [Print Replica] Kindle Edition (2014),Publisher: Pearson Education; 8 edition (September 4, 2014)
3. Costan A. “From Big Data to Fast Data: Efficient Stream Data Management. Distributed, Parallel, and Cluster Computing [cs.DC]”. ENS Rennes, 2019. https://hal.archives-ouvertes.fr/tel-02059437v2/document
4. Mozafari, B., Zeng, K., D’Antoni, L., and Zaniolo, C. “High-performance complex event processing over hierarchical data”. ACM Trans. Datab. Syst. 38, 4, Article 21 (November 2013), 39 pages. DOI: http://dx.doi.org/10.1145/2536779
5. XQuery 1.0.” An XML Query Language” www.w3.org/TR/xquery/
6. Gray P.M.D. “AMOSQL In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston”, MA, (2009)
7. Fahl G., Risch T., and Sköld M.”1AMOS – an architecture for active mediators”. In Proc. Workshop on Next Generation Information Technologies and Systems, 1993
8. Hassan SolimanHazem M El-BakryHazem M El-BakryMona Reda. “Real-time transmission of video streaming over computer networks”, February 2012 ,Conference: Proceedings of the 11th WSEAS international conference on Electronics, Hardware, Wireless and Optical Communications, and proceedings .
9. Asif A. N. , Samiul I.T., Motahar T. .” Automated Mobile Robot with RFID Scanner and Self Obstacle Avoiding System. January 2018,Conference: International Conference on Inventive Computing Systems and Applications (ICICSA 2018),At: Pattaya, Thiland
10. Hatem H., Motaz S., Ramzi A .” Performance Evaluation of RESTful Web Services for Mobile Devices”, International Arab Journal of e-Technology, Vol. 1, No. 3, January 2010
11. Forrester Consulting. “Real-Time Data Analytics Empowering Publishers to Make Better, Faster Decisions”, 2015.
12. Jimmy D.”Data-Intensive Text Processing with MapReduce”, University of Maryland, College Park Manuscript prepared April 11, 2010
    Donald M. and Adam S. “MapReduce Design Patterns”, Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.2013
13. Rabin D. “Big Data and Google File System”, August 2016,Conference: 8th National Students' Conference on Information TechnologyAt: Kathmandu, Nepal
14. M. Tamer Özsu ,Patrick Valduriez .”Principles of Distributed Database Systems” Third Edition. Springer New York Dordrecht Heidelberg London ISBN 978-1-4419-8833-1 e-ISBN 978-1-4419-8834-8 ,DOI 10.1007/978-1-4419-8834-8. 2011
15. Jonathan Leibiusky, Gabriel Eisbruch, and Dario Simonassi ,”Getting Started with Storm”, Printed in the United States of America. Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
16. Justin E. “Hadoop, Storm, Samza, Spark, and Flink: Big Data Frameworks Compared”, digitalocean , 2016.
17. Douglas C. “JavaScript: The Good Parts”, Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. May 2008
18. Saurabh Zunke, Veronica D’Souza. “JSON vs XML: A Comparative Performance Analysis of Data Exchange Formats” ,2014, IJCSN International Journal of Computer Science and Network, Volume 3, Issue 4, August 2014 ISSN (Online): 2277-5420
19. Nurzhan Nurseitov Michael Paulson Randall Reynolds Clemente Izurieta . “Comparison of JSON and XML data interchange formats: A case study”. January 2009 SourceDBLP Conference: Proceedings of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering, CAINE 2009, November 4-6, 2009.
20. Branislava ŠandrihDušan TošićVladimir Filipović .” Towards Efficient and Unified XML/JSON Conversion - a New Conversion Method. Transactions on Internet Research (TIR)”, 13(1):58–64, January 2017.
21. Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein,Wei Hong, Sailesh Krishnamurthy, Sam Madden,Vijayshankar Raman, Fred Reiss, and Mehul Shah.”TelegraphCQ: Continuous Dataflow Processing for an Uncertain World+”.University of California, Berkeley,Intel Berkeley Laboratory IBM Almaden Research Center 2003.
22. Boumedyen Shannaq, AlShamsi,I.Saif,N,”Management Information System for Predicting Quantity Martials”.TEM Journal, 8(4), 1143-1149, DOI: 10.18421/TEM84-06.(2019).
23. Al-Azzawi,F. & Boumedyen Shannaq. “Fuzzy Analysis Model for Classifying Exams Questions in Learning Quality Management System Based on Bloom’s Taxonomy Verbs”. 4. 69-79. 10.4206/aus.2019.n26.4.9, (2019).

AUTHORS PROFILE

Dr.Boumedyen Shannaq is a skilled university professor for 10 years, primarily teaching Information Systems and Information Technology Courses. Interested in Electronic Learning, Innovation, Artificial Intelligence, Analysing Big Data. Professional skills in: Public Speaking, Time Management, Self-motivation, Inter-personal communication and Record keeping ,Focussed on Analyze ideas and use logic to determine their strengths and weaknesses.Manage oneself, people, time, and things.Obtained PhD in Information Systems (data Engineering-Technical Issues).

.