iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 8 **Issue:** VI **Month of publication:** June 2020

DOI: http://doi.org/10.22214/ijraset.2020.6350

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue VI June 2020- Available at www.ijraset.com

Data Streaming Process using Elasticity Strategy: A Survey

R. Kokila[1], Dr. S. Radha Rammohan[2]
[1]*Dr. MGR Educational and Research Institute, India*
[2]*M. Sc., M. Tech., Ph.D., MIE(India), CE(IEI)*

Abstract: ***This paper investigates reactive elasticity model in stream process environments the performance goal is to investigate massive amounts of data records with low latency and minimum resources. Operating in the context of Apache Storm, the system proposes an elastic management strategy that modulates the similarity degree of applications' elements while probably addressing the hierarchy of execution containers (virtual machines, processes and threads). The system shows that provisioning the incorrect related container that may cause performance degradation and propose a resolution that provisions the minimal expensive container to extend performance. The system describes the observation metrics and show that the process takes under consideration the specifics of execution surroundings. The system provides an experimental analysis with real-world applications that validates the relevancy of the approach. Further, the system is applied in real time execution where it contains DDos Analysis and online data analysis avoiding the attacks and traffics.***
Keywords: Data streaming, elasticity strategies, apache storm

I. INTRODUCTION

Big data is considered as a major challenge in varied automatic data processing system domains. it' is provided in IoT with the proliferation of connected devices, grows with the increasing scale of high performance computing systems and is including the increasing net and social network activities. Also it is a major topic within the data intelligence business. There are two major techniques to handle massive data: Batch processing and Stream processing.

BATCH PROCESSING: In batch processing, information is initially saved in large databases and is processed later, sometimes with scalable programming models like Google's MapReduce [1]. However, with the ever growing size of data, the price of data transfer and storage becomes preventative [2], [3]. Moreover, in multiple domains, what is necessary isn't to stay the initial data however to research it as quick as attainable to provide valuable intelligence..

STREAM PROCESSING: To tackle these problems, stream process systems place the stress on reactivity and analyze data because it is created. Recent years have seen the emergence of multiple stream process solutions [4,5] .

The performance issues of static provisioning have impelled the utilization of elasticity at times when resources are allotted on-demand. If existing proposals share a similar goal on minimizing resource consumption whereas maintaining application performance, they change in their responses to why, when and the way to scale resources. In terms of performance goals (the why question), existing solutions pursue cost effectiveness or aim at implementing QoS guarantees on applications' latency [6], [7] and throughput. To choose when to adapt the amount of allotted resources, solutions either try and protect performance issues through predictive application models, or react to performance degradation using application observation.

Stream process systems (SPSs) method provides boundless streams of input tuples by evaluating them consistent with a given set of queries. This processing model permits to interrupt down complicated computations into simple units (the operators), independently position them, and deploy the ending system over any variety of computing machines. Having the computation performed in parallel by many distinct operators on several machines is that the core feature of distributed stream process systems. Such flexibility permits to scale horizontally in such a simpler way to produce the computational power needed to sustain a given tuple input load with affordable process latency. Due to these characteristics, SPSs nowadays represent a basic building block for a maximum amount of huge data computing infrastructures. A hard challenge SPSs have to be compelled to deal with is input dynamism. Such systems, in fact, are designed to ingest information from heterogeneous and probably intense sources like sensor networks, observing systems, social feeds, etc. that are usually characterized by massive fluctuations within the input file rates. Solutions related to over-provisioning are taken cost in-effective during a world that moves toward on-demand resource provisioning designed on top of IaaS platforms.

In [10], Spark Streaming outperforms Storm in peak throughput measurements and resource utilization but Storm attains lower latencies. The Yahoo benchmark confirmed that under high throughput Storm struggles. Karakaya et al. [9] extended the Yahoo Benchmark and found that Flink 0.10.1 outperforms Spark Streaming 1.5.0 and Storm 0.10.0 for data-intensive applications. However, in applications where high latency is acceptable, Spark outperforms Flink. Karimov et al. [8] confirmed that the general

performance of Spark Streaming 2.0.1 and Flink 1.1.3 is betterthan that of Storm. In this work, we include the latest versions of Spark Streaming 2.4.1 and Flink 1.9.1 due to their superior results. Additionally, we include Spark's new Structured Streaming 2.4.1 API and Kafka Streams 2.1.0, which have not been thoroughly benchmarked before [11].

Giselle van Dongen et al., 2020 The paper analyze the relationship between latency, throughput, and resource consumption and the system measure the performance impact of adding different common operations to the pipeline. To ensure correct latency measurements, the process use a single Kafka broker. The results show that the latency disadvantages of employing a micro-batch system are most apparent for unsettled operations. With a lot of advanced pipelines, custom implementations may provide event-driven frameworks an oversized latency advantage. because of its micro-batch design, Structured Streaming can handle terribly high throughput at the value of high latency. below tight latency SLAs, Flink sustains the high level throughput. in addition, Flink shows the smallest amount performance degradation once confronted with periodic bursts of data [13].

Shilpa Chaturvedi,[14] et al., 2019 declared a "Dataflow utilize algorithms" that once given a submitted dataflow, determine the intersection of reusable tasks and streams from existing dataflows to create a merged dataflow, with secure equivalence of their output streams. Algorithms to unmerge dataflows once they are removed, and defragment partly reused dataflows are used. The system implement these algorithms for the Storm fast-data platform, and validate their performance and resource savings with use of 86 real and artificial dataflows from eScience and IoT domains.

RUICHANG LI, CHUNKAI WANG et al., 2020 [15] "This paper presents the partitioning framework RCDC (Runtime Correlation Discovery) consistent with runtime correlation discovery. RCDC implements the total granularity partitioning strategy, which incorporates runtime direct correlation partitioning (RPC-partitioning) and group partitioning (Clu-partitioning). First, in the method of RPC-partitioning, the author introduces the mini-batch theme to minimize the amount of output stream caches and partition data streams using the connection of partitioning keys. moreover, in the method of Clu-partitioning, the system re-partition data streams by grouping of inclined data streams between the inter-node and therefore the intra-node. Then, the method construct the routing table to manage partition states so as to confirm the correctness of multiple query tasks. Finally, the system enforced this framework on Apache Storm".

Benjamin Heintz et al., 2017 The author represent a family of optimum offline algorithms that conjointly minimize these metrics, and therefore the system use these to guide the architecture of sensible on-line algorithms related to the insight that windowed classified aggregation may be designed as a caching issue wherever the cache size varies over time. more the system evaluates the algorithms through associate implementation in Apache Storm deployed on PlanetLab. By using workloads derived from anonymzed traces of a well-liked analytics service from an oversized commercial CDN, the paper experiments show that the net algorithms accomplish near-optimal traffic and staleness for a range of system configurations, stream arrival rates, and queries [16].

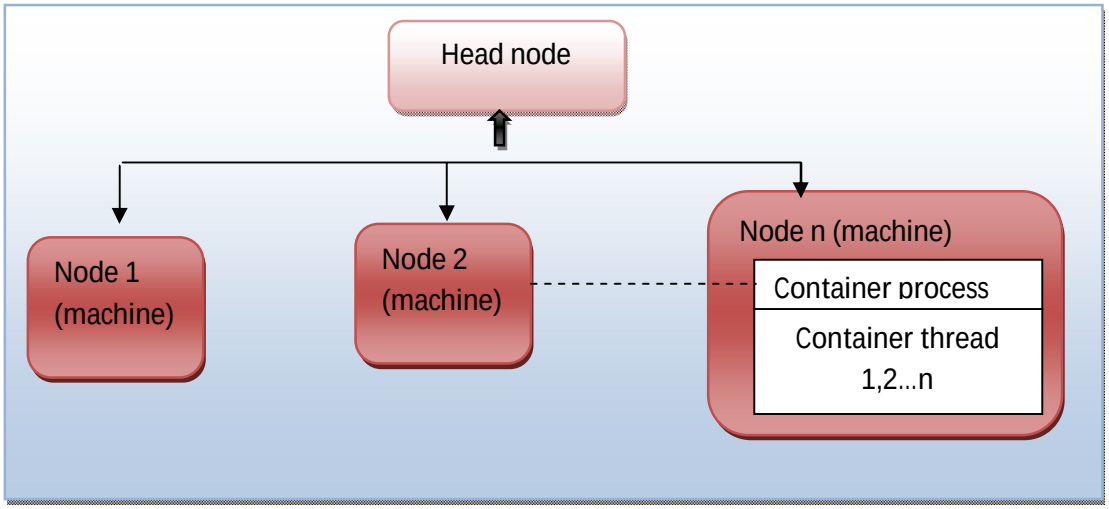
II. METHODOLOGY

A. Elasticity for Data Stream Processing

Distributed stream system frameworks are designed to perform continuous computation on probably endless data streams whose rates may vary over time. On account of making solutions to form such systems elastic scale is a basic goal to attain desired performance and detach prices caused by resource over-provisioning [18]. These systems are often scaled on particular two divisions: the operator similarity and therefore the variety of resources. In this survey paper, the system show the view of these two dimensions, on dependent entities, are independent however should simultaneously act for the worldwide advantage of the system. On the premise of this observation, the system proposes a fine-grained model for estimating the resource utilization of a stream process application that permits the independent scaling of operators and resources. A simple and effective combined management of the two dimensions permits us to propose multi-level elasticity a previous concept, a unique elastic scaling approach that gives effective resource utilization. the system enforce the approach among Apache Storm and tested it by running the real-world applications with totally different input load curves [19]. The outcomes segregate the claims showing that the proposed dependent management outperforms elasticity methods wherever operators and resources are conjointly scaled and measured.

1) *Working:* The study tackles the elasticity process issue the various methods round: the process further target the execution atmosphere and investigate how modifying the amount of various execution containers impacts application performance. The essential plan is that completely different execution containers have different capacities that goes with different costs. Provisioning a better capacity container, usually a Virtual Machine is more expensive, in resources and time, than provisioning a lower-capacity one, corresponding to a method or a thread. However, a thread that exists within the context of a machine and can't be issued of there are not any accessible resources. The goal is to mechanically provision the minimal expensive resources

capable of satisfying the applications' performance desires. At process hierarchy of containers, further resolution leverages the resources of higher-capacity containers using lower-capacity container consolidation.



B. Multi-level Elastic Resource Provisioning.

the system propose a basic elastic strategy that considers various levels of execution containers. The intuitive plan is to provision the minimum range of significant containers (e.g. VMs or processes) and to higher resource usage by multiplying the amount of light-weight containers (threads). More significantly, the strategy accounts for the performance impact of various execution containers as provisioning the incorrect form of resource can really decrease performance. the method devise an easy however effective strategy to benchmark applications and to experimentally discover the least (cheapest) configuration for a given work.

C. Transparency Integration into Apache Storm.

The method implements the proposal in Apache Storm, the next level Apache project supported by an energetic community as well as massive corporate like Twitter and Yahoo. the implementation preserves the Storm API and provides for provisioning of execution containers at various levels. the implementation mechanically benchmark applications, discover the least configurations to support a given work and generate elasticity controllers to manage application bottlenecks. The system additionally validates the approach with two real-world applications cases.

1) First is industrial DDoS application process real workload.

2) Second is an online data analysis of results created by the CoMD simulation application.

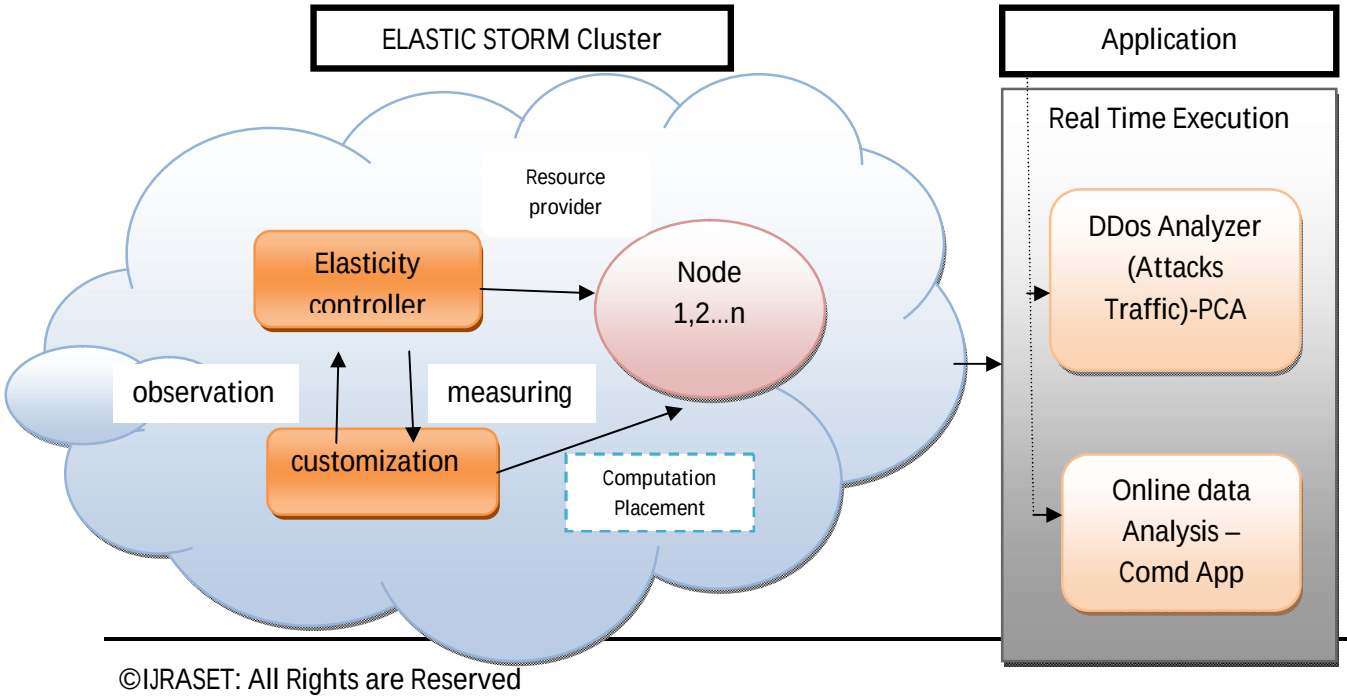


Fig 2. Elastic Storm when applied in real time application

3) The method the DDoS application used to distinguish between normal and abnormal network traffic. It uses the PCA- principal Component Analysis a statistical method in order to learn about the changing system behavior and is able to detect even never seen before attacks.

4) The elasticity controller puts together resource management, application monitoring and application scaling. For resource management, the method considers the deployment of Storm on a cloud platform and uses the available resource provisioning features.

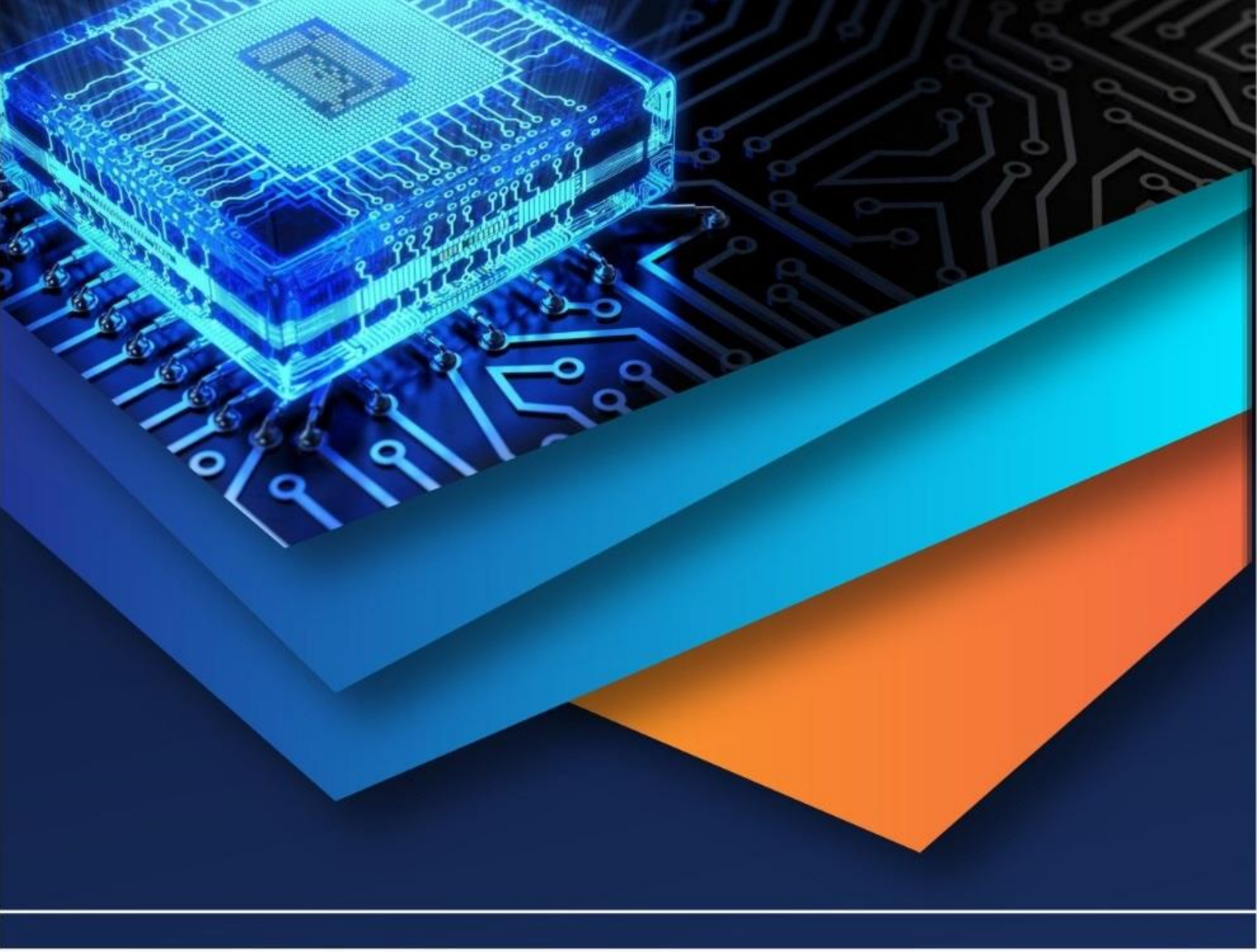
5) When provisioning new nodes, the controller configures them with the appropriate number of workers (one) and inserts them into the Storm cluster.

III. CONCLUSION

The focus of the paper is on the impact of various execution containers on the performance of an elastic stream process system. The paper has taken the hierarchy of executing containers (machines, processes and threads) and has shown that their provisioning comes at a distinct cost. More significantly, the elastic method showed that provisioning the incorrect form of containers that could decrease performance. The system further defines an elastic resource management for the Apache Storm system that scales an application whereas using the most cost effective resource configuration. The implementation is transparent to applications because it preserves the Storm API and a transparency model. As shown with two real-world applications, the system succeeds in maintaining the end-to-end latency of applications by detection and healing local bottlenecks. The least execution configurations to support given levels of employment are discovered at a preliminary section of application bench marking. This approach is simple however economical in reflecting both the particular execution surroundings and application desires. In our future work the method poses in integration specific models of any preliminary data regarding system.

REFERENCES

[1] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Communications of the ACM, vol. 51, no. 1,pp. 107–113, 2008
[2] D. Duellmann. (2015) Big data and storage management at thelarge hard on collider.
[3] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma, “In situ visualizationfor large-scale combustion simulations,” Computer Graphicsand Applications, IEEE, vol. 30, no. 3, pp. 45–57, May 2010.
[4] R. . Young, “Big data: changing the way businesses competeand operate,” http://www.ey.com/Publication/vwLUAssets/EY__Big_data:_changing_the_way_businesses_operate/\$FILE/EY-Insights-on-GRC-Big-data.pdf, 2018.
[5] Oracle, “Oracle fast data: Real-time strategies for big dataand business analytics,” http://www.oracle.com/us/solutions/fastdata/fast-data-gets-real-time-wp-1927038.pdf, 2013.
[6] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica,“Discretized streams: Fault-tolerant Streaming Computation atScale,” in Proceedings of the Twenty-Fourth ACM Symposium onOperating Systems Principles. ACM, 2013, pp. 423–438.
[7] B. Lohrmann, P. Janacik, and O. Kao, “Elastic stream processing with latency guarantees,” in 35th IEEE International Conference on Distributed Computing Systems, ICDCS 2015, Columbus, OH, USA, June 29 - July 2, 2015, 2015, pp. 399–410.
[8] J. Karimov, T. Rabl,A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl, “Benchmarking distributed stream processing engines,”in Proc. IEEE 34th Int. Conf. Data Eng., 2018.
[9] Z. Karakaya, A. Yazici, and M. Alayyoub, “A comparison of stream processing frameworks,” in Proc. Int. Conf. Comput. Appl., 2017, pp. 1–12.
[10] S. Chintapalli et al., “Benchmarking streaming computation engines:Storm, flink and spark streaming,” in Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops, 2016, pp. 1789–1792.
[11] R. Lu, G.Wu, B. Xie, and J.Hu, “Streambench: Towards benchmarking modern distributed stream computing frameworks,” in Proc. IEEE/ACM7th Int. Conf. Utility Cloud Comput., 2014, pp. 69–78.
[12] S. Qian, G. Wu, J. Huang, and T. Das, “Benchmarking modern distributed streaming platforms,” in Proc. IEEE Int. Conf. Ind. Technol., 2016, pp. 592–598.
[13] Giselle van Dongen , Member, IEEE and Dirk Van den Poel. “Evaluation of Stream Processing Frameworks” 2020.
[14] Shilpa Chaturvedi, Student Member, IEEE, Sahil Tyagi, Yogesh Simmhan, Cost-effective Sharing of StreamingDataflows for IoT Applications IEEE Transactions on Cloud Computing 2019.
[15] RUICHANG LI, CHUNKAI WANG and FAN LIAO “RCDC: A Partitioning Method for DataStreams Based on Multiple Queries” 2020.
[16] Benjamin Heintz, Member, IEEE, Abhishek Chandra, “Optimizing Timeliness and Cost in Geo-Distributed Streaming Analytics” 2017.
[17] P. Huber, Robust Statistics, ser. Applied Probability and Statistics Section Series. Wiley, 2004.
[18] I. Safieddine, “Optimisation d’Infrastructures de Cloud Computing dans des Green Datacenters,” Ph.D. dissertation, UGA, 2015.
[19] F. Lombardi, L. Aniello, S. Bonomi, and L. Querzoni, “Elastic symbiotic scaling of operators and resources in stream processing systems,” IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 3, pp. 572–585, March 2018.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY