Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

Adaptive Scheduler in Apache Storm

Dr. Jayalakshmi D S
Associate Professor,
Dept. of CSE RIT, Bangalore

Manaswini J S
Student at M S Ramaiah Institute of Technology
Dept.of CSE RIT, Bangalore

Dr. Geetha J
Associate professor
Dept.of CSE RIT, Bangalore

Abstract:- Apache Storm is a distributed, error-tolerant, and highly scalable platform for streaming data. Storm supports a wide range of use cases, including real-time analysis, machine learning, continuous computation, and much more. This is extremely fast, with the ability to check one million records per second for a medium-sized cluster node. In the Storm framework, the Storm default scheduler distributes a topology workload equally into worker processes using a simple round-robin algorithm without considering any priority or criteria, leading to higher response time and less throughput. Two advanced general scheduling proposals provide better performance for the storm. The network connection works by analyzing the topology and traffic and load constraints between nodes RST scheduling works; the second scheduling enhancement is the former method by redistributing executors through available slots at runtime to improve overall performance.

Keywords: Apache Storm, round-robin Throughput, Constraints

1. INTRODUCTION

The rapid development of social networks, IoT (Internet of Things), and cloud computing have entered the world due to society's big data age. Process command in the Big Data Stream The team has two types of processing methods that are manageable. Stream processing enables real-time analytics, such as video surveillance, analysis in e-commerce and social networks, compared to batch processing. Currently, many systems stream process, among them Apache Storm, is adopted by many popular companies, e.g., Twitter.

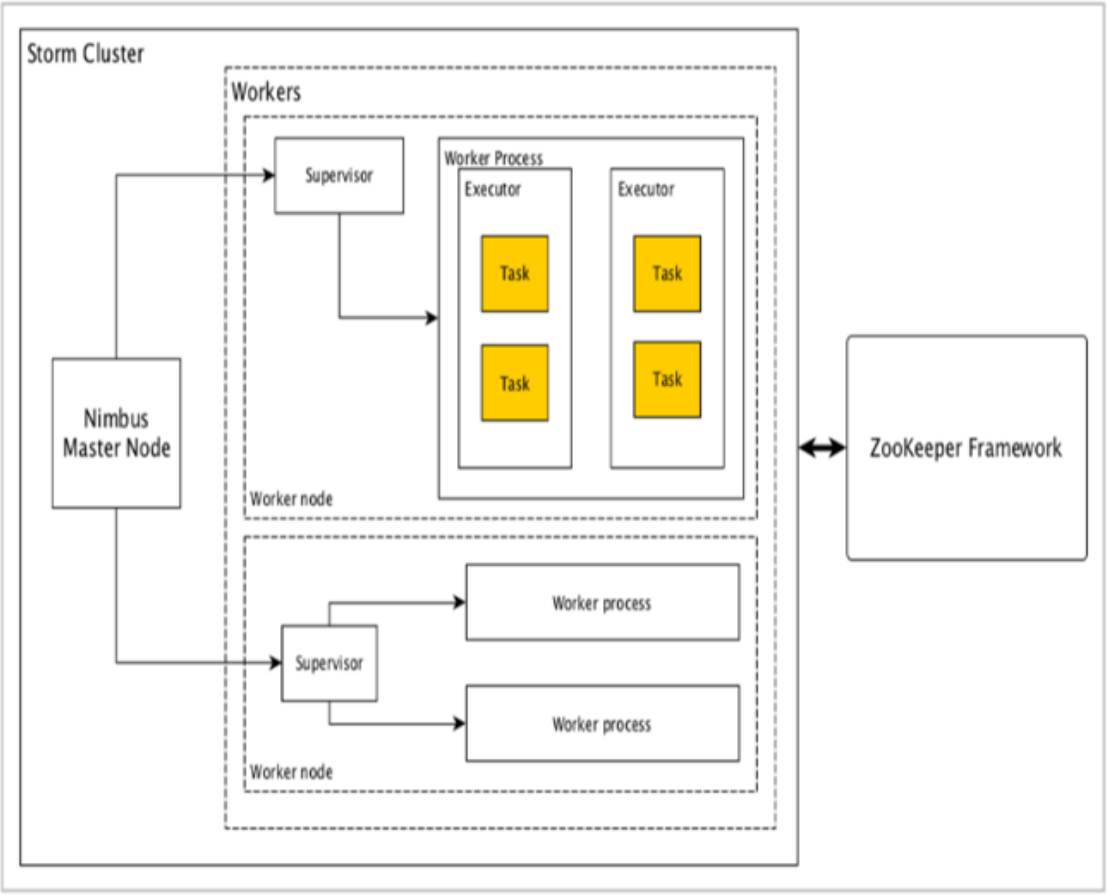

Figure 1. Storm Architecture

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

Apache Storm framework is a real-time, distributed big data-processing system that means Storm is a framework that distributes stream processing for real time computations. Storm continuously processes a vast amount of data even when one of its components fails. Apache Storm consists of two types of nodes, and one is the master node, which is Nimbus, and the other is the worker node or slave node, which is Supervisor. Figure 1 shows Storm cluster architecture. The central component of Apache Storm is Nimbus. Nimbus runs storm topology. Nimbus takes the topology and analyses the task to be executed. Then, it distributes the task to Supervisor. Supervisors assign tasks to worker processes. The worker process will run the task on available executors.

Apache Storm is stateless. This behavior of Storm helps to process real-time data in the fastest way. Apache ZooKeeper helps to store Storms state. A failed Nimbus can be restarted since the state is available in Apache ZooKeeper and continues to work from where it left. This section describes the issues of the current scheduling strategy of Apache Storm. Storm uses a round-robin approach, a default scheduling strategy that allocates executors to all available slots evenly. And also, in a single slot, no executors from various topologies will exist. This leads to a longer response time and less throughput.

The user's pre-configuration determines the number of executors and worker numbers in the topology when applying the Storm cluster. These configurations can be modified to get better utilization of resources and performance. But it isn't easy to specify appropriate parameters without evaluating the state of the cluster. Default scheduler does not track load status such as internode traffic, CPU usage. It does not analyze the topology structure also.

Additionally, improper usage of workers can harm the performance. Thus, we can see that run time analysis is gaining importance for better performance of the scheduler. Therefore, the adaptive scheduler is needed to optimize the configuration of topologies and task assignments. The main objective of this work is to provide an improvised apache scheduler than a default scheduler. Following are the objectives of the work:

i. Implement a basic scheduler which analyses the submitted topology structure and distributes task to executors, executors to slots and finally slots to nodes.
ii. Implement the rescheduling strategy based on the executors' run time data by optimizing the topology and task assignment configurations.
iii. Evaluate the implemented scheduler under different loads.
iv. Test the proposed model with different topologies.

2. LITERATURE SURVEY

Nowadays, Big Data has made its place as the most demanded technology in the market. To handle big data we have many software that can increase the efficiency. The data volume is generated and increased with demands of increasing series of sensing devices in IOT. It is known as big data: heavy data with different sets and it’s grown in speed that avoids the traditional processing and use of solutions [1]. People pose technological hurdles such as space and memory limits, latency and bandwidth considerations, fault tolerance when processing IOT data series, i.e., data streams from many users and devices [2]. A stream of data continuously processed in real time is provided solutions for several problems called data stream processing (DSP) [3].

The DSP implementation is distributed in ways where several nodes can separate the problem into smaller one [4]. Apache Storm [5], an distributed stream processing engine (SPE) is used to run these distributed stream processing applications (SPAs). For analyzing the case it has been posted on different social media such as Twitter. Map Reduce [6], a Google product is most widely used programming based on Hadoop [7], open source implementation and made available to others outside that company. Stream processing includes data processing before storing and like batch systems involves processing after storing. Generally, a processing unit in a stream engine is called a processing element[8]. A stable scheduling [9] is more critical than an efficient scheduling for streaming applications, especially when a scheduling is to be dynamically rescheduled during runtime [10-11].

A stable scheduling [9] is more critical than an efficient scheduling for streaming applications, especially when a scheduling is to be dynamically rescheduled during runtime. There are many other personalized schedulers, except regular Storm scheduler such as the online T-Storm [13] scheduler. Based on reducing inter-slot communication cost, T-Storm's scheduling strategy is to minimize the total communication costs. However, when the cluster load is heavily loaded or the nodes in a data center are distributed multiple racks, the inter-node communication is far more expensive.

3. PROPOSED ARCHITECTURE DESIGN

The layered implementation of Apache Storm framework and the architectural components are as shown in in Figure 2. The lowest layer is the Hardware layer that consists of peripheral devices such as CPU, memory like RAM, HDD.

Next layer is the operating system layer. Operating system layer itself can be divided into multiple layers. Its main functionalities are process management, CPU scheduling, system monitoring, interaction with I/O devices, memory management. And user programs and applications like browsers, processors run in this layer. Apache Storm also run in this layer. The next layer is Storm server layer. Actual scheduler is implemented in this layer. The functionalities like Data manager, Node manager, Traffic monitor, Load monitor, Task monitor, Log manager etc which are required for scheduling are implemented in this layer. The top-most layer is the User Code that exposes configuration related functionalities for Storm (Storm Zookeeper servers, Ui ports, Nimbus host, Supervisor slot ports), user requirements and scheduling policies. At this layer user can perform tests based on the configurations.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

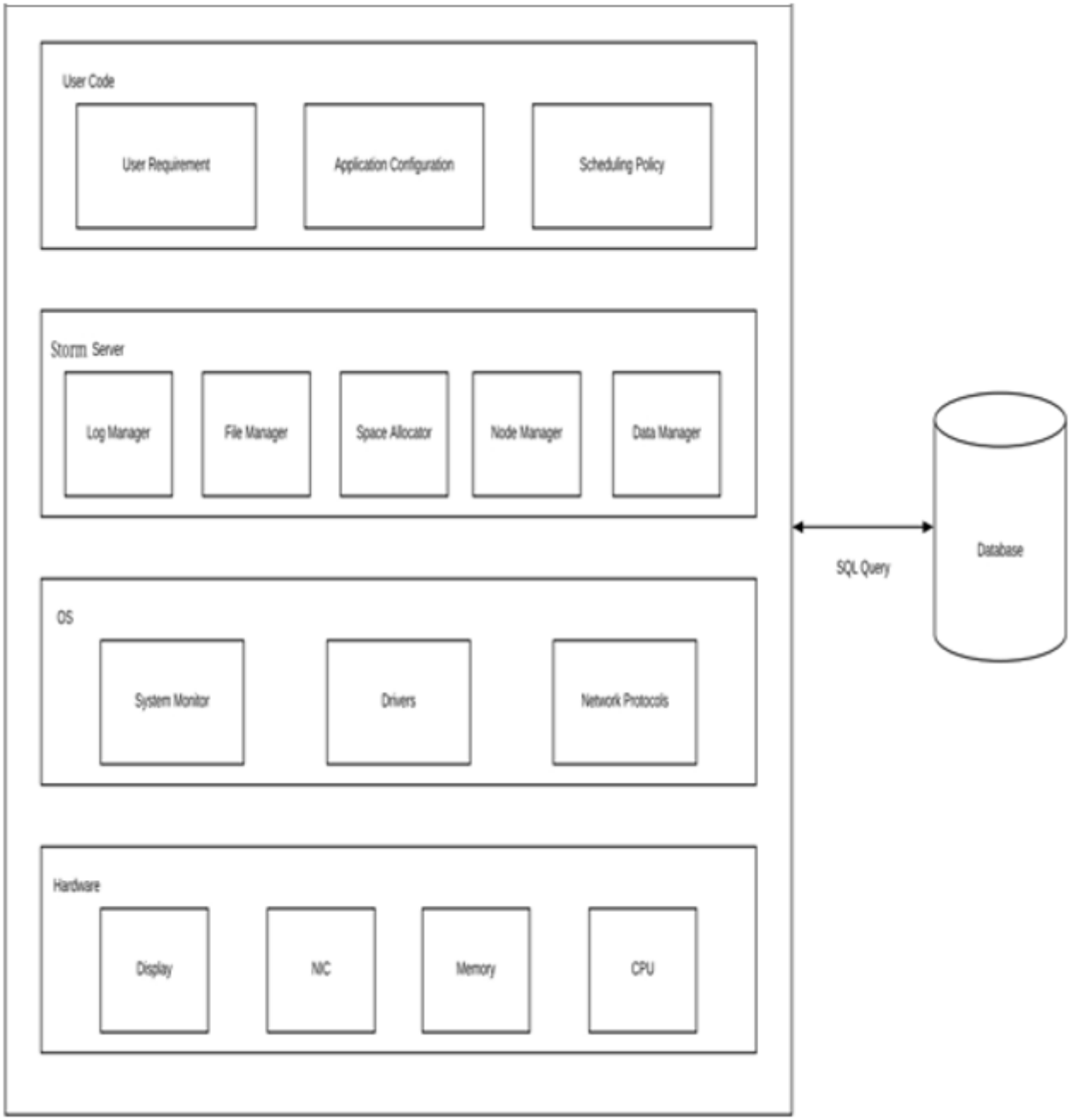

Figure 2. Layered Architecture Diagram

System Architecture identifies underlying modules of the system. It is very important to identify the modules at the early stages of development and recognize the communication between the already established modules. The functionalities of each module are defined clearly.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

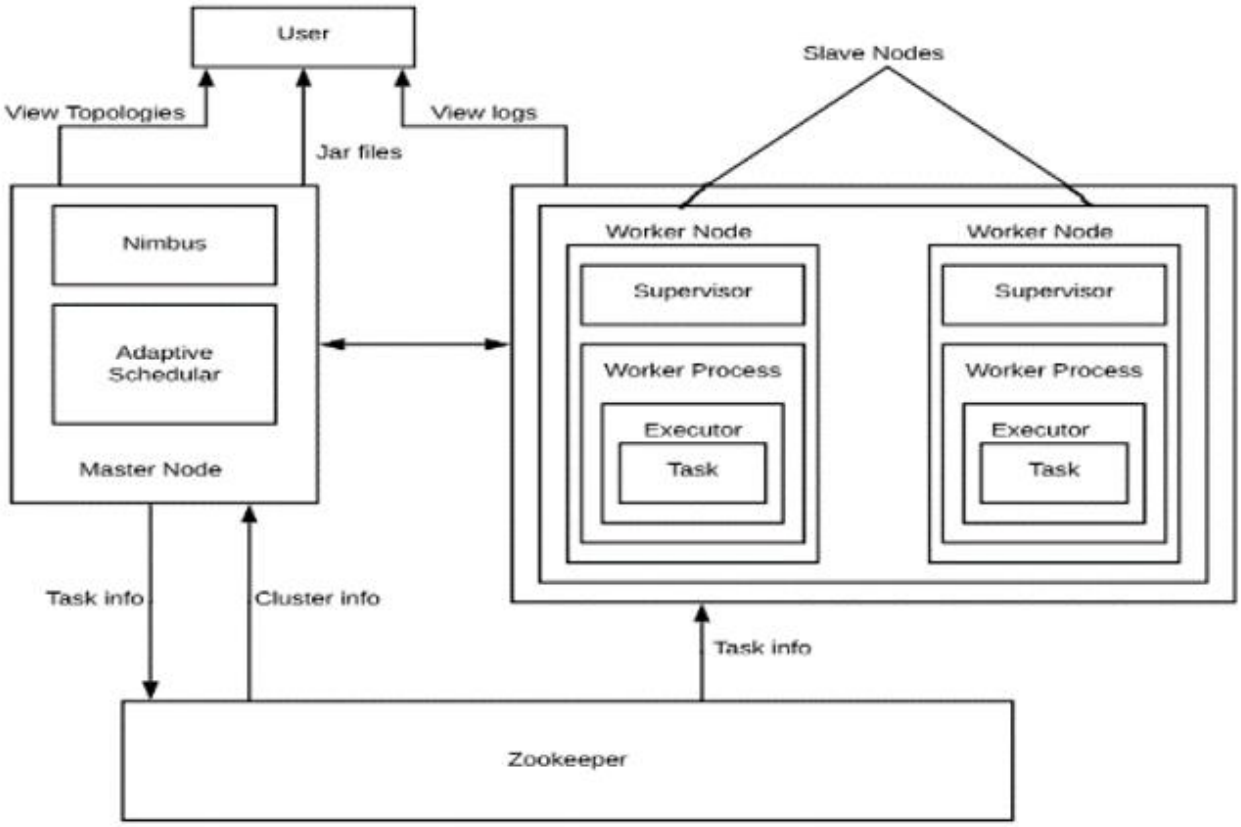

Figure 3. Architecture Diagram

Figure 3 explains the architecture of the work. Topologies are submitted to master node from applications like twitter. Nimbus is the master node where scheduling strategy is implemented. Zookeeper is responsible to manage state of Nimbus as Storm is stateless. If Nimbus dies, it can be restarted from where it stopped because Zookeeper stores all states. Once the topology is submitted Nimbus analyses its structure and distributes tasks to all available Supervisors. After particular time interval Nimbus checks for rescheduling criteria and reassigns tasks. Once all tasks completed Supervisor waits for new task. Once all topologies are processed Nimbus also waits for new topology similar to Supervisor.

3.1 Class Diagram

In this section the overview of the high-level classes and the relation between them is presented. The important classes in the Apache Storm scheduler framework package relevant to work are shown in the Figure 4. The classes such as Data manger, Node manager, and Inter Node Traffic manager manage topology, node, and traffic. The classes Load monitor, Task monitor, Work monitor are used to monitor load, task and works. Scheduler class is used to schedule the tasks based on designed criteria.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

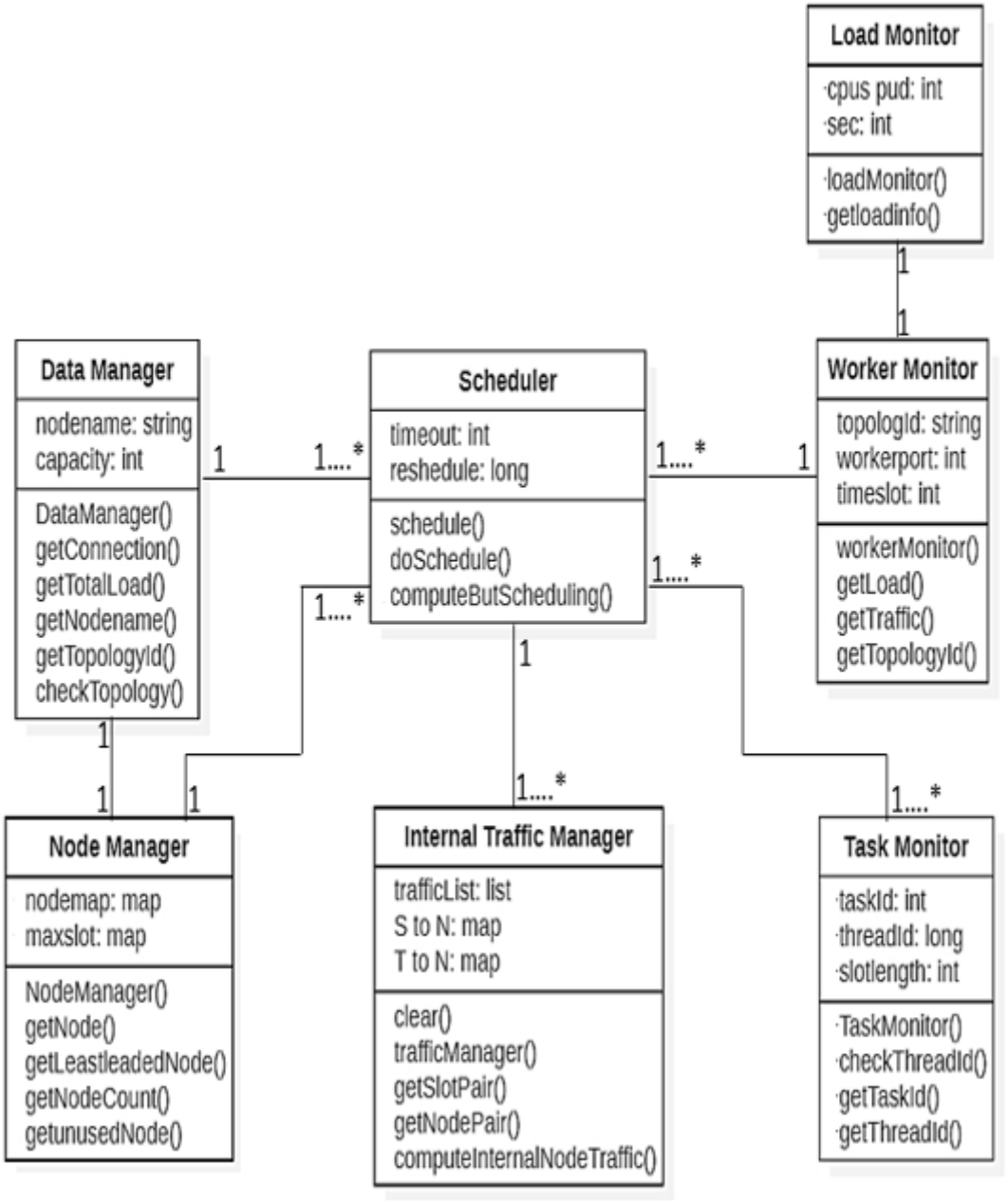

Figure 4. Class Diagram

3.2 Data Flow Diagrams

Data flow diagrams are the diagrammatical representation of the flow of data through various components. It can be illustrated as a pipeline function where the data flow from its origin, performs all required computations and finally reach the endpoint. The last stage is the output produced by that module. Figure 5 represents the flow of data between the different processes of the scheduling method. The topology obtained from application is provided to topology structure analysis module. This data will be processed and assigns executors and slots. Based on the internode traffic rescheduling happens. Tasks are submitted to reschedule module and finally processed data is generated as output.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

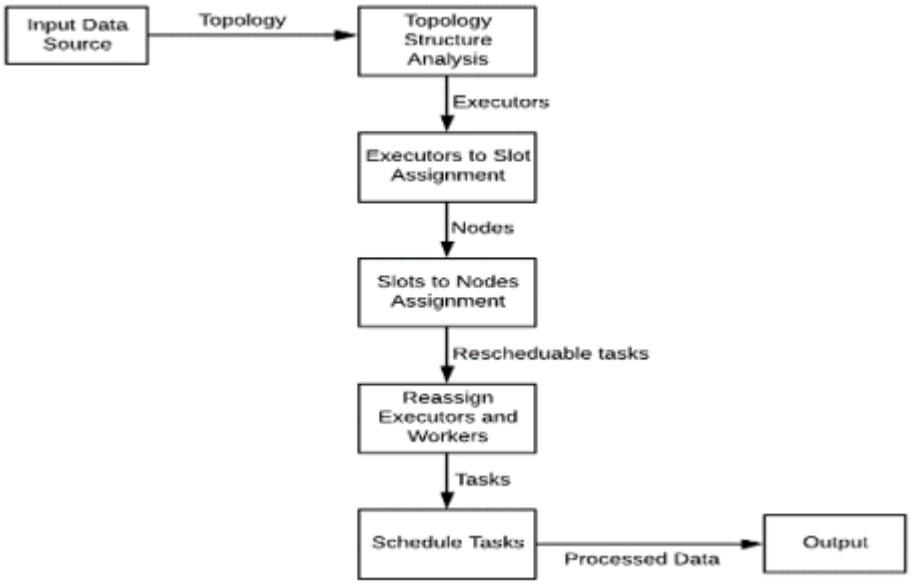

Figure 5 Data Flow Diagram

3.3 Activity Diagram

Dynamic aspects of the system are elaborated in the activity diagram. Activity here is the basic operation of the system. Various operations of the system are divided into different activities. The Activity diagram acts as a flowchart which displays the flow of control between these activities.

Figure 6 Activity Diagram

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

As shown in the Figure 6, first activity is to get topology from application. Then analyse its structure and schedule tasks. If reschedule is required reassign executors and workers. Otherwise continue with previous scheduler.

3.4 Sequence Diagrams

The sequence diagram portrays the interaction between the different entities of the system. The Figure 7 shows the high-level modules and the sequence of communication between them. The internal working of the scheduler is also shown in the diagram.

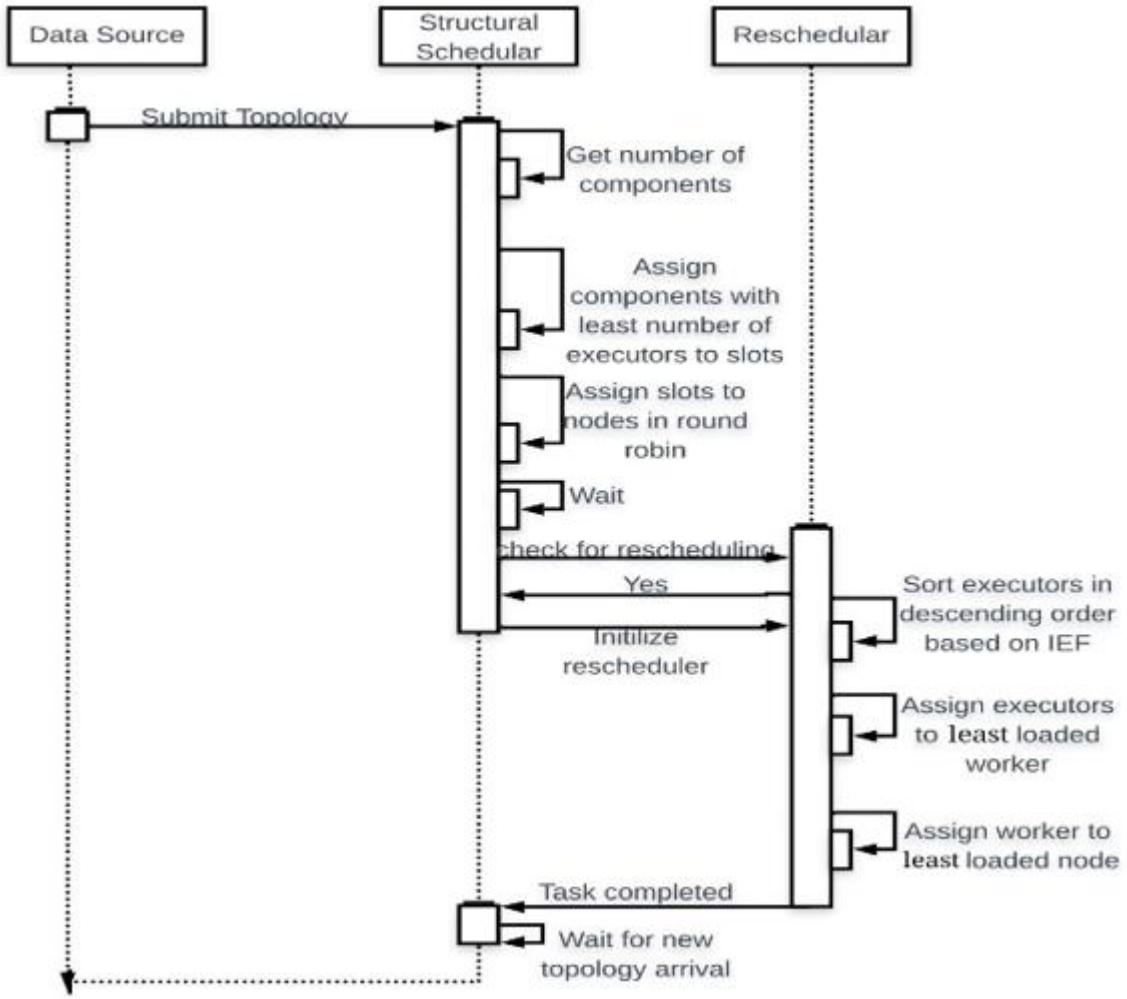

Figure 7 Sequence Diagram

3.5 Explanation of the Algorithm and how it is been Implemented

A rapid process required to carry out Rebalance at runtime indicates that some discovery needs to be used to identify a new distribution. The following algorithm is based on a basic greedy quest from the load balancing node across handlers, which means that inter-node traffic is reduced and all nodes can be avoided. It consists mainly of two successive stages.

Each topology-splitting operator is split into what workers have been designed to serve as the topology stage. The placement goal is to reduce the traffic between different workers' managers and balance the CPU demand of the workers. In the second phase, the first phase's workers must be assigned to cluster's available slots. To minimize inter-node traffic and to satisfy load capacity constraints such allocation has taken into account.

3.5.1 Executor to worker assignment

Algorithm : ETW(T, E, N, W, $L_{i,j}$, $R_{i,j,k}$)

Input: T-set of topologies, N-set of nodes, E-set of executors, W-set of workers, L-load generated by executor $e_{i,j}$, R-tuple rate between executors.

Output: Execution Path to be implemented.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

Begin
Foreach *topology* $t_i \in T$ do
IET ← $\{(e_{i,j};e_{i,k};R_{i,j,k})\}$
Sorted descending order $R_{i,j,k}$
Get least loaded worker
$W^* \leftarrow \text{argmin}_{wi,x \in W} \sum A1(e_{i,y}) = w_{i,x}\, L_{i,y}$
If !assigned($e_{i,j}$) and !assigned($e_{i,k}$) then
$A1(e_{i,j}) \leftarrow w^*$
$A1(e_{i,k}) \leftarrow w^*$
Else
Check the best assignment to workers that already include either executor and to w*.
End

3.5.2 Worker to node assignment

Algorithm : WTN(T, E, N, W, $L_{i,j}$, $R_{i,j,k}$)
Input: T-set of topologies, N-set of nodes, E-set of executors, W-set of workers, L-load generated by executor $e_{i,j}$, R-tuple rate between executors.
Output: Execution Path to be implemented.
Begin
Foreach *topology* $t_i \in T$ do
IST ← $\{(w_{i,x};w_{i,y};\gamma_{i,x,j})\}$
Sorted descending order $\gamma_{i,x,j}$
Get least loaded node
$n^* \leftarrow \text{argmin}_{n \in N} \sum A2(e_{i,y}) = n\, L_{i,y}$
If !assigned($w_{i,x}$) and !assigned($w_{i,y}$) then
$A1(w_{i,x}) \leftarrow n^*$
$A1(w_{i,y}) \leftarrow n^*$
Else
Check the best assignment to nodes that already include either worker and to n*.
End

In the first phase, the pairs of interacting executors are iterated for each topology in descending order by rate of interchanged tuples. If both the executors have not yet been allocated for each of these pairs, they are assigned to the least loaded worker. Similarly, in the second phase, the interacting workers' pairs are iterated by rate of exchanged tuples in descending order. If both have not been allocated to any node yet for each pair, then the least loaded node is selected to host them.

4. RESULTS AND DISCUSSION

All the testing was performed in Eclipse IDE. As the work provides an extension to Apache Storm, it is tested in the Apache Storm environment where our work is built.

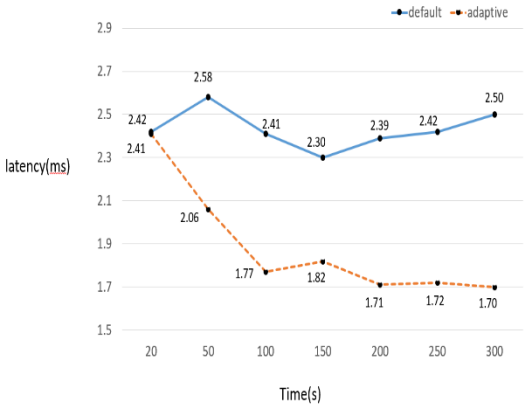

Figure 8: Latency v/s time

The figure 8 shows Latency v/s time for default and adaptive schedulers. At the beginning system seems overloaded and all schedulers experiences a transient state. This period lasts upto 20 seconds. In the subsequent 20 seconds time frame (up to second 40), both schedulers' performance is stable. Adaptive scheduler stores admin actions used between 40 and 50 seconds after they start to recompile. At second 50 adaptive scheduler starts to work and quickly the performance is improved than default scheduler. This proves that the adaptive scheduler analyzes run time behavior and correctly identify cases to improve performance.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

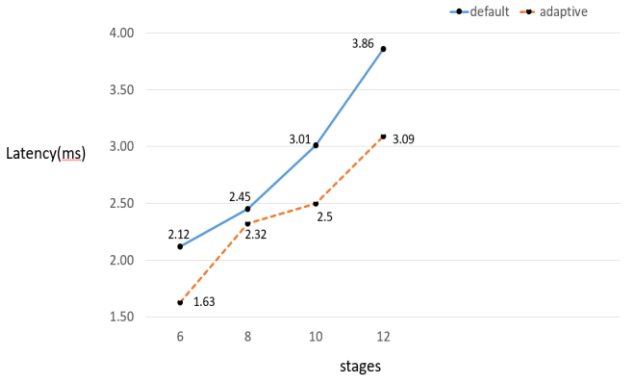

Figure 9: Latency v/s number of stages for default and adaptive schedulers with repliction factor 2 for each stage

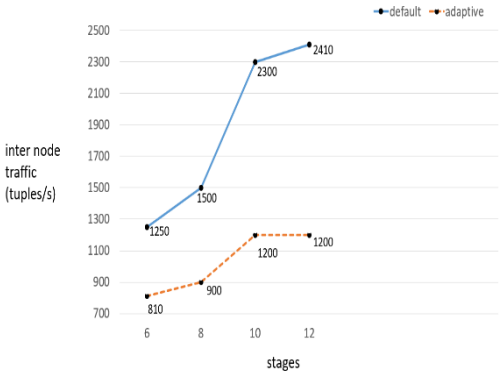

Figure 10 Inter-node traffic v/s number of stages for default and adaptive schedulers with repliction factor 2 for each stage

Inference from figure 9 and figure 10: 2 executors are assigned on each component. Latencies of adaptive scheduler is less compared to the default scheduler. The results on inter-node traffic reflects the adjustable scheduler's default low-spaced node traffic. Latency and inter-node traffic are not always directly related.

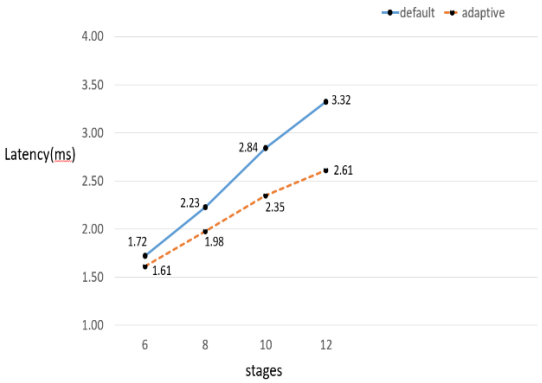

Figure 11: Latency v/s number of stages for default and adaptive schedulers with repliction factor 4 for each stage

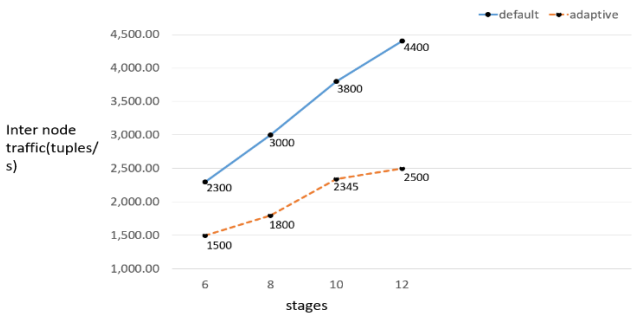

Figure 12: Inter-node traffic v/s number of stages for default and adaptive schedulers with repliction factor 4 for each stage

Inference from figure 11 and figure 12: The adaptive scheduler provides lower latencies than the default one. 4 executors are assigned on each component. Adaptive scheduler generates lower inter-node traffic than default one.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

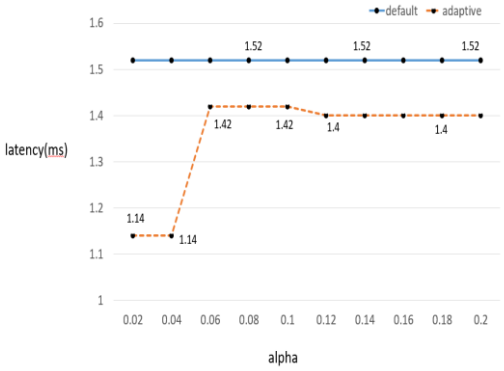

Figure 13: Latency v/s alpha for default and adaptive schedulers

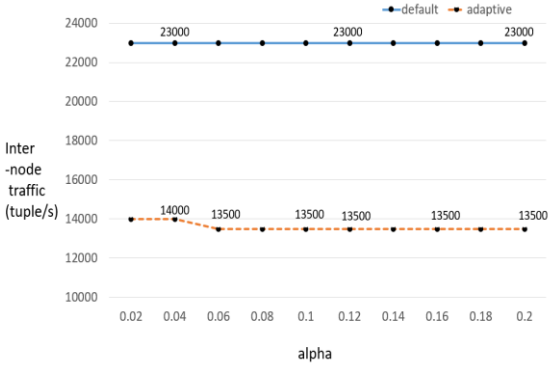

Figure 14: Internode traffic v/s alpha for default and adaptive schedulers

Inference from figure 13 and figure 14: 5 executors are assigned on each component i,e total 30 executors and Alpha varies from 0 to 0.2. Adjustable scheduler provides its performance when α 0.05. Increases latency but keeps inter-node traffic low.

5. CONCLUSION

Big data storm is an emerging technology and its open source loop with increasing real-time employment, assisting, and developing it. Storm is expected to support a wide range of use cases and its associated complexity, making the storm's development difficult and the new features of the drive itself to adapt to almost all use cases. In this work, the default scheduling of existing scheduling problems is first analyzed. Storm scheduling, adapting its behavior according to the topology and application of the midget time communication mode is designed and implemented. Experimental results show that developed scheduling achieves better performance in terms of waiting time than the default storm scheduling in production. The latency of processing events is less than 15-25% regarding the test topology's default storm scheduling. Following are the few enhancements which can be done as future work in this work:

i. Need to enhance the sorting technique for executors with inter-node traffic to take less time for execution.
ii. Topology requires more configurations and parameters before submitting to the scheduler. The work can be extended to reduce topology dependency.
iii. The work main consideration is latency and internode traffic. This can be extended to throughput to process more data/tuples per unit time.

REFERENCES

[1] A. McAfee and E. Brynjolfsson, “Big Data: The Management Revolution,” Harvard Business Review, 2012.
[2] H. Andrade, B. Gedik, and D. Turaga, Fundamentals of Stream Processing. Cambridge University Press, 2014.
[3] M. Stonebraker, U. C¸ etintemel, and S. Zdonik, “The 8 Requirements of Real-Time Stream Processing,” ACM SIGMOD Record, 2005.
[4] C. P. Chen and C.-Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on Big Data,” Information Sciences, 2014.
[5] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, “Storm @Twitter,” in 2014 ACM SIGMOD Int. Conf. on Management of Data.
[6] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in Proc. Int. Symp. Operating Syst. Des. Impl., 2004.
[7] Apache Hadoop Work, Oct. 2016. [Online]. Available: http://hadoop.apache.org/
[8] Survey of Distributed Stream Processing for Large Stream Sources,Supun Kamburugamuve, Prof. Geoffrey Fox, Prof. David Leake, 2013.
[9] A Stable Online Scheduling Strategy for Real-Time Stream Computing Over Fluctuating Big Data Streams. School of Information Engineering, China University of Geosciences, Beijing 100083, China Corresponding author: D. Sun.
[10] Performance Modeling and Predictive Scheduling for Distributed Stream Data Processing. Teng Li, Student Member, IEEE, Jian Tang, Senior Member, IEEE, and Jielong Xu.
[11] A Hybrid Scheduling Algorithm based on Self-Timed and Periodic Scheduling for Embedded Streaming Applications. Amira Dkhil, Xuan Khanh Do, St´ephane Louise CEA, LIST ,PC172, 91191 Gif-sur-Yvette, France. Christine Rochange IRIT, University´e de Toulouse 118 route de Narbonne, Toulouse, France.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 02, February-2021

[12] L. Aniello, R. Baldoni, and L. Querzoni, "Adaptive online scheduling in Storm," in Proceedings of the 7th ACM international conference on Distributed event-based systems. ACM, 2013.
[13] J. Xu, Z. Chen, J. Tang, and S. Su, "T-Storm: traffic-aware online scheduling in Storm," in Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on. IEEE, 2014.
[14] Priority-based Resource Scheduling in Distributed Stream Processing Systems for Big Data Applications. Paolo Bellavista, Antonio Corradi, Andrea Reale, and Nicola Ticca Department of Computer Science and Engineering.
[15] Iterative Scheduling for Distributed Stream Processing Systems. Leila Eskandari, Jason Mair, Zhiyi Huang, David Eyers Department of Computer Science University of Otago Dunedin, New Zealand.
[16] S-Storm: A Slot-aware Scheduling Strategy for Even Scheduler in Storm. Wenjun Qian, Qingni Shen, Jia Qin, Dong Yang, Yahui Yang and Zhonghai Wu School of Software and Microelectronics, Peking University, Beijing 100871, China[2016].
[17] Adaptive Task Scheduling in Storm, Jiahua Fan, Haopeng Chen, and Fei Hu, School of Electronic Information and Electrical Engineering Shanghai Jiao Tong University, China[2015].
[18] N-Storm: Efficient Thread-Level Task Migration in Apache Storm, Zhou Zhang, Peiquan Jin, Xiaoliang Wang, Ruicheng Liu, Shouhong Wan, School of Computer Science and Technology, University of Science and Technology of China [2019].
[19] The Real-time Scheduling Strategy Based on Traffic and Load Balancing in Storm, Jing Zhang, Chunlin Li, Liye Zhu, Yanpei Liu, Department of Computer Science, Wuhan University of Technology, Wuhan China[2016]