QoS-Aware Self-Adapting Resource Utilisation Framework for Distributed Stream Management Systems

Tarjana Yagnik
Computing, Engineering and Media
De Montfort University
Leicester, United Kingdom
email: tarjana.yagnik@dmu.ac.uk

Feng Chen
Computing, Engineering and Media
De Montfort University
Leicester, United Kingdom
email: fengchen@dmu.ac.uk

Laleh Kasraian
Computing, Engineering and Media
De Montfort University
Leicester, United Kingdom
emai: laleh.kasraian@dmu.ac.uk

***Abstract*— The last decade witnessed plenty of Big Data processing and applications including the utilisation of machine learning algorithms and techniques. Such data need to be analysed under specific Quality of Service (QoS) constraints for certain critical applications. Many frameworks have been proposed for QoS management and resource allocation for the various Distributed Stream Management Systems (DSMS), but lack the capability of dynamic adaptation to fluctuations in input data rates. This paper presents a novel QoS-Aware, Self-Adaptive, Resource Utilisation framework, which utilises instantaneous reactions with proactive actions. This research focuses on the load monitoring and analysis parts of the framework. By applying real-time analytics on performance and QoS metrics, the predictive models can assist in adjusting resource allocation strategies. The experiments were conducted to collect the various metrics and analyse them to reduce their dimensions and identify the most influential ones regarding the QoS and resource allocation schemes.**

***Keywords*—Data stream management; Distributed stream processing; quality of service; resource allocation; prediction; scheduling.**

I. INTRODUCTION

During the last decade, a new category of data-intensive systems and applications has emerged and has been recognised by the researchers and industry professionals in the data science field. A data stream is defined as a real-time, continuous, ordered sequence of data items [1].

The systems deployed to manage data streams are called Data Stream Management Systems (DSMS) [2]. A common definition of such system is that it is the system especially developed and assembled to process continuous queries on dynamic and ever-changing data streams. DSMSs are totally different from the traditional Database Management Systems (DBMSs) in that traditional database management systems expect the data to be persistent in the system and the queries to be dynamic while within the DSMSs paradigm it is expected to have dynamic unbounded data streams and the queries are submitted on those streams as persistent queries.

Quality of Service (QoS) [3] is identified as an important attribute of overall performance measure of any system. One of the main challenges within the QoS management of DSMS systems, is how to efficiently and effectively deliver predefined QoS requirements. Within a system that has multiple queries submitted over different data streams, different queries would have totally different QoS requirements and constraints.

The DSMS should have the ability to distribute and allocate physical computing resources between those queries and fulfil the required QoS specification in a fair and square manner. The DSMS utilises a mechanism called *scheduling strategy* to allocate the available resources based on the various queries' QoS requirements.

There are two main issues [4] that have to be dealt with when managing resource allocation of distributed data stream systems and they are:

- The ability of the system to allocate or release computing resources to meet an application workload and specified quality of service requirements; and
- Devising and performing the relevant optimisation actions to alter the system configuration during the runtime to utilise any additional capacity or release previously allocated resources to guarantee the agreed-upon (or specified) end-to-end quality of service levels of the system critical applications.

A framework for QoS-Aware Self-Adaptive Resource Utilisation management of data stream management systems is presented in this paper. QoS is tightly connected between all system components, i.e., each component within the system contributes to the overall quality of service perceived by the system applications. A comprehensive usage model that comprises mixing *instantaneous reactions* with *proactive actions* is incorporated within the proposed framework. Instantaneous reactions include applying real-time analytics on collected performance and QoS metrics of each component of the system (worker profiling) before such data becomes obsolete and loses its value. Proactive actions are the processes of applying predictive models that further assist in decision making and resource allocation planning and scheduling within the system in a real-time streaming environment.

This paper is organised as follows. Section II gives a background and a quick overview of the current research activities and proposed QoS-aware, resource utilisation and scheduling frameworks in the field. Section III introduces the proposed framework and gives an overview of the architecture and the different components within the framework. Section IV explains the various performance metrics and QoS metrics that will be utilised as part of the framework. Section V

 ISBN: 978-1-61208-842-6

gives an overview of the Apache Storm, its terminologies and techniques as a case study of the experimental verification of the proposed framework. Section VI details the experiments that have been conducted to collect the various metrics and analyse them to reduce their dimensions and identify the most influential ones. The paper is concluded in Section VII and planned future work is also outlined.

II. Background

Although there have been so many papers and projects that have researched and discussed the various performance aspects in DSMSs [5]–[8], many of those papers had concentrated their attention on the study of the different individual components within the DSMS. There has been an evident lack of studying and investigating the QoS of the whole DSMS system.

A. Resource-Aware and Traffic-Aware Scheduling Schemes

A great deal of research [9]–[13] has been carried over the last decade focusing on scheduling of streaming applications. We present in this section a comparison between the most related resource- and traffic-aware schedulers from different perspectives. Some of the traditional schemes proposed for resource allocation and scheduling [12] rely on measuring a set of performance metrics and consequently make some adjustment to the scheduling strategy or resource allocation schemes based on a comparison with a pre-defined set of constrains and measurements thresholds.

Those schemes suffer from the lack of dynamic adaptability to the real-time and timely-constrained fluctuations in the system workload and data input patterns. Schedulers within DSMSs can be categorised into two classes:

- Static schedulers, such as the default scheduler in Storm [13], where executers are assigned as evenly as possible between all workers and tasks are assigned on a round robin fashion to different task executers. Another type of static schedulers is the resource-aware scheduler called R-Storm [8], proposed by Peng et. al. and implemented in the latest versions of Storm (v2.0 and beyond).
- Dynamic schedulers, where the scheduler plan or strategy is adjusted during run-time and the system configuration is changed to reflect the main scheduler goal or/and quality of service requirements of the system. These schedulers can be classified into the following three categories:
  - Throughput oriented schedulers, as being presented in [15] and [16].
  - Latency-oriented schedulers, as those presented in [17] [18] and
  - Communication reduction schedulers, as the three schedulers presented in [14] [19] [20].

B. Scheduling Optimisation Approaches

The scheduling optimisation approaches within DSMS platforms can be classified based on the main objectives of its scheduling strategy. They are either built based on minimising or maximising CPU utilisation, memory usage, throughput or satisfy certain pre-defined quality-of-service levels. The various data streaming scheduling strategies can be classified based on their scheduling objectives.

1) Minimising Memory Consumption: The memory consumption within a DSMS depends mainly on the size of the operator's buffer in addition to the current internal state of each operator. The first approach is vital to the process of transferring data tuples from each operator to the next one. The second approach is governed by the number of tuples that the operator needs to fulfil its data grouping requirements(join, aggregation, duplicate elimination, etc.). The scheduler prioritise its placement of the operator based on the ability of the operator to reduces the amount of data exchanged and processed within the shortest period of time. This strategy has been the core scheduling strategy in Aurora [21] and STREAM [22] systems and is called Min-Memory and Greedy approaches, respectively.
2) Minimising CPU Utilisation: Reduction in the number of calls to an operator results in the reduction of CPU usage and consequently reduce the overhead associated with the scheduling activities. Examples of such approaches are the Aurora's super-boxes and tuple trains [21] as implemented within the QStream's micro period method [23].
3) Maximising Quality of Service: Maximum QoS metrics are mainly reflected within the DSMS system with their ability to minimise the total output delay and with maximising the system overall throughput. Both methods are considered below:
   - Minimising Delay: This strategy is corner stoned with the well-known First In First-Out (FIFO) allocation strategy. The optimisation objective is fulfilled by pushing or pulling certain data tuples through the network of operators as fast as possible. As a result of such optimisation, the delay time (complete tuple latency) is shortened so it fells within the acceptable range specified by the query submitter or the system application.
   - Maximising Throughput: Whenever a scheduling strategy aims at minimising the overall system CPU utilisation, it automatically enables the system to attain a higher output throughput under a given resource availability.
4) Scheduling for Specified Level of Quality-of-Service: In order to guarantee certain levels of quality of service for critical applications such as health monitoring system, critical infrastructure applications, military command/control applications, it is vital to the proper operation of the system to ensure that such strict QoS levels are always met and provided to such application under various operation conditions. A minimum level of throughput can be guaranteed by certain DSMS platforms so it has to ensure the system will not cross a maximum output delay limits.

 ISBN: 978-1-61208-842-6

C. QoS and Resource-Aware DSMS Frameworks

There are many DSMS that have emerged during the last decade. Apache Storm [13] is one of the most popular systems in this area. It had attracted both industries and researchers' attention from early stages of its development. Aurora [21] and QStream [23] are amongst the main DSMS frameworks that incorporate some levels of QoS assurance from within. A limitation in those systems is that the QoS specifications are provided only for the output streams. Other approaches, such as Borealis [23] and the one proposed by Klein and Lehner [24], manage QoS specifications at the operator level.

D. Self-Adapting Approaches

The approach presented in [25] is a self-adaptive, resource management framework for software services in the Cloud. This approach utilises historical data to build a prediction model to predict the QoS value of a certain management operation. Serhani et. al. [26] proposed a layered-architecture, self-adapting framework that supports end-to-end workflow management with the ability to adapt its configuration and quality specifications and enforcement mechanisms. It uses a declarative, rule-based cloud services orchestration approach to detect event patterns and utilise machine learning algorithms to build a meta-model for monitoring and adaptation of workflow within cloud services. The approach differs from our proposal by focusing into application-based and content-based end-to-end QoS attributes. Cardellinin et. al. [27] proposed a framework that aimed at extending Apache Storm to operate within FOG computing environment. This approach mainly focuses on QoS attributes related to the geographically distributed network environment.

Our approach differs substantially from those approaches by looking deeply into individual component's time-based QoS and performance metrics, analysing those metrics to reduce their dimensions in order to use them to build a dynamic prediction model through incremental learning algorithms along with ensemble learning and abnormality detection.

III. The Proposed Framework

Throughout this paper, a QoS-Aware self-adapting resource allocation framework is proposed. It enables the collection of dynamic performance and QoS metrics for each component within the DSMS submitted query plan or topology. This work contributes to the growing interest in introducing QoS considerations in the data streams domain and helps in supporting further classes of QoS-sensitive streaming applications at scale.

A. Framework Architecture

Figure 1 shows the system architecture of our proposed framework. The framework is composed of four main components: QoS Management, Performance Metrics Management, Incremental Learning Prediction with Abnormality Detection, and the Dynamic Tuning/Scheduler Adapter Module. The following subsections elaborate on the structure, functionality and overall processes involved in each component

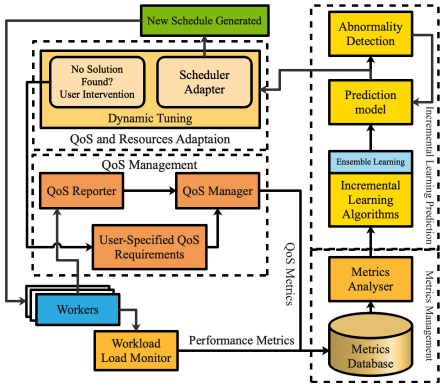

Figure 1. Framework Architectural design.

1) QoS Manager: The QoS Manager stores the various reports that it receives from its reporters located within the various components within the DSMS. For a given QoS constraint, the manager will keep all measurement data concerning the monitored topology component element during the current measurement window span (t) time units and consequently will discard the older data measurements. The QoS management is composed of the following two elements:
   - User QoS Specifications: Each application user should specify the QoS constraints that are needed to be fulfilled during the queries execution or computations over the input streams. This can be provided, through a high-level abstracted QoS interface, as QoS graphs or QoS tables regarding any of the main QoS metrics.
   - System QoS Metrics Collector: The master node within the cluster has global knowledge about each one of the pre-defined QoS requirements. It will inform each worker about where and when QoS metrics measurements should be collected. This will minimise the needed resources to collect such metrics and the computing power needed to analyse them on the fly so the computing overhead of this component is minimised. For every query data operator with user-specified QoS constraints, QoS metrics will be measured once during the configured time window called the measurement interval.
2) Performance Metrics Management: All performance metrics are measured and associated calculation are done over the specified window of time and then the aggregate results will be reported through a metrics collection pipeline to the Metrics Management Module. The module runs certain metrics normalisation, factor and correlation analysis to reduce the data dimensions and identify the most relevant set of metrics that can be fed to the next component which is the Incremental Learning module.
3) Incremental Learning Prediction with Abnormality Detection: The DSMS cluster nodes are affected by fluctuations that may be caused by the abnormal data rates,

 ISBN: 978-1-61208-842-6

network transmission and other factors, or the fact that some of the measured metrics and QoS observations don't fit into the prediction model.

4) Dynamic Tuning and Scheduler Adapter Module: The Scheduler Adapter is the component entrusted in making those decisions. The Scheduler adapter initiates necessary countermeasures in the form of modifications to the run-time scheduling strategy and system global configuration of the deployed cluster and worker node settings, until the specified QoS constraint has been met or there is no additional measures can be made in order to meet such constraint. In this case the result will be reported to the user to decide on the best action to be done (either by adding additional physical resources or relax the QoS specifications for this particular query).

B. Implementation Tools and Techniques

In order to validate the proposed framework and evaluate its performance, the following tools are being used to facilitate the development of a prototype that proves the concept and validate the effectiveness of the proposed framework. This includes the following in-house developed systems, open-source tools and related techniques:

1) Apache STORM.
2) QoS Management Module.
3) Performance Metrics Management Module.
4) WEKA [28] and MOA [29] ML Toolkits.
5) Performance Dynamic Tuning Module.

IV. Data Streams Metrics

The reporting of the metrics can be in the form of gauges, histograms and numeric values. Often these result in multiple metrics being uploaded to the reporting system, such as percentiles for a histogram, or rates for a meter.

A. Performance Metrics:

Throughout our experiments, the Metrics Management module collects the following metrics:

1) Data-Level Metrics: Two main metrics related to the stream data input are:
   - *Data Input Rate*: The rate of input data stream can be controlled by using the Data-Rate metric, by changing the spout parallelism with the Apache Storm cluster. The data rate of a data stream describes how many stream tuples per second occur in this stream.
   - *Data Delay*: Along with the Data Rate, a delay metric is defined as the time interval from a certain tuple arrives at the first input component of the topology and the time it leaves the time it is inserted to the next stream within the system.
2) Query-Plan/Topology Metrics: These metrics are related to the queuing behaviour of the system specially those related to how much time a tuple is setting waiting to be processes by the next component. Sample of the metrics within this category are:
   - *Window-Size* and *Sliding-Step* metrics: Those metrics denote the total number of input and output tuples within a sepecified time measurement window.
   - *Op-Selectivity*: This metric is defined as the total number of data tuples that are emitted relative to the number of data tuples consumed by the topology/query plan operator. For example, if an operator emits 2 tuples as a result of consuming one input tuple, then the Op-Selectivity of that particular operator is 2.
3) Scheduler-Level Metrics: The objectives of a distributed stream management scheduling algorithms is to maximise throughput and system resource utilisation and minimising the latency while trying to meet the user/application requirements and quality of service constraints. The *Uptime metric* measures the total computation time that is consumed by a running Java Virtual Machine process within the worker node.
4) Cluster Level Metrics: Apache storm version 2.2 provides an extensive set of cluster metrics which include the following categories:
   - *Cluster Metrics*: These are metrics that are reported through the nimbus daemon with metrics that report the state of the cluster and its various components.
   - *Supervisor Metrics*: Metrics associated with the supervisor, which launches the workers for a topology.
   - *UI Topology Metrics*: Metrics associated with a topology running in the cluster and reported through a single UI daemon. The metrics can be collected through the extensive REST UI API interface within the DSMS.

B. Quality of Service Metrics:

Within the context of this research, there are several parameters that are able to represent both the query submitter's performance satisfaction, as well as the system's throughput in a variety of scenarios. The experiments are designed in a way that takes into consideration the following most relevant QoS metrics. Those metrics give clear picture and understanding about the effective factors that are taken into consideration in regard to QoS and scheduling optimisation approaches as shown in Table I.

V. Apache Storm

Apache Storm is an open-source, real-time, scalable distributed and fault-tolerant Data Stream Processing System maintained by the Apache Software Foundation. On a physical Storm cluster deployment, there are several types of entities used to execute a topology as shown in Figure 2 and they are:

1) Task: It denotes an instance of a certain topology component (Spout or Bolt) or query plan operator.
2) Executor: which is used to execute one or more tasks related to the same operator.
3) Worker or Work Process: which runs one or more executers on the same topology.

 ISBN: 978-1-61208-842-6

TABLE I
QUALITY-OF-SERVICE (QOS) METRICS

Metric	How to Measure It
Total Through-put	This metric can be measured by calculating the total number of tuples processed within the specified amount of time called measurement window.
Memory Consump-tion	It is measured by the amount of memory in MB that is consumed by a certain component to execute its functions of the query plan as part of the whole memory consumption of the system.
CPU Util-isation	This metric is defined as the amount of CPU resources consumed by a certain query plan component where it is available on a certain task/executor and is represented by a point system.
Complete Latency	This denotes the average time for a tuple tree to be completely processed from end to end by the topology compoennets.
System Latency	It is measured by the amount of time the system takes to process each tuple or it denotes the time difference between the tuple input and its output from the system.
Processing Latency	This metric is measured by the time the system or any component within the system to process the tuple it receives and output it to the upstream for output or further processing.
Execution Latency	It can be measured by calculating the delay in time experienced by the system when it completely execute the tuple or tuple tree.

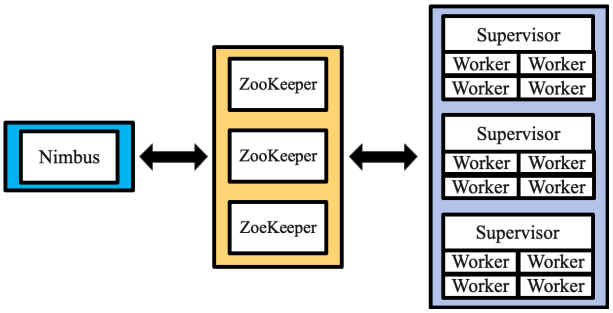

Figure 2. The various componenets and daemons within the Apache Storm Cluster Setup.

4) Worker node: It denotes a physical computational resource or simply a computer instance.
5) Zookeeper: a server application (Daemon) that is responsible for managing the cluster configuration parameters and enable distributed coordination between the different cluster nodes and processes.
6) Nimbus: which is the centralised management "brain" of the cluster and charged with the overall topology execution.
7) Supervisor: It is responsible for starting or terminating worker processes based on the Nimbus assignment and keep coordination with Nimbus for any fault-tolerance mechanisms to implement in case of node-failure

The processing of data streams within Storm is delegated to several types of components within the platform. Each component is responsible for executing simple tasks. The component within the Storm cluster that handles incoming streams of data is called Spout. The Spout function is to pass streams to another component called Bolt. Bolts are usually used to transform streaming data in different ways. The Bolt may store the data in certain form of storage or process it and pass it to another Bolt. The Storm cluster can be viewed as a chain of spouts and bolts arranged in a certain connections layout to form a Direct Acyclic Graph (DAG) called Topology.

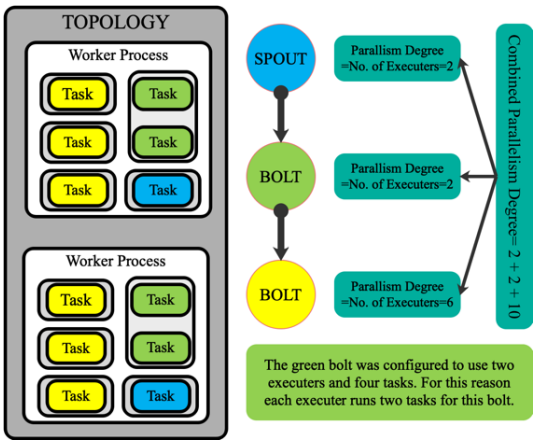

Figure 3. Mapping the logical layout of the WordCountTopology into physical worker nodes, worker tasks and executors within the Storm Cluster.

The illustration in Figure 3 shows how a simple topology would look like in operation within the Apache Storm Data Stream Processing System. One of the widely-used benchmark topologies within Storm is the WordCountTopology shown in Figure 4. This topology has been used to measure the various performance metrics and speed of messaging between spouts and bolts. The topology is composed of the following three simple components:

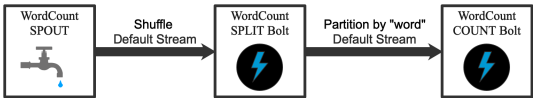

Figure 4. The logical layout of the WordCountTopology.

1) Spout: The *SPOUT* component within a topology will simply emit a stream of data tuples under the name "sentence" as a sequence of string value. To simplify the implementation of the experiment, the input data tuples are retrieved from a static list of sentences that is accessed repeatedly based on the input rate settings of the spout. The spout will repeat a loop of generating and then emitting one tuple for every sentence. In the real-world scenarios, such process is replaced by a data producing mechanism and sentences are received through certain APIs such as Twitter API or through topics producers as used in Apache Kafka or similar data ingestion platforms.
2) Split Bolt: The *SPLIT* bolt subscribes to the "default" stream of sentences emitted by the SPOUT. For each tuple received from the spout in the form of a sentence (stream of strings), it splits the sentence into words, and

 ISBN: 978-1-61208-842-6

emit a tuple for each word to the upward stream of the next connected component of the topology. It uses different stream grouping mechanisms available within Storm to deal with streaming to multiple components or join multiple streams into one main stream.

3) Count Bolt: The *COUNT* bolt subscribes to the output stream (also called default stream) coming from the split bolt. Its main function is to count how many times it has seen a particular word. Every time this bolt receives a tuple from the input stream, it will increment its counters accordingly.

VI. EXPERIMENTS

In order to establish a solid baseline for our research, several experiments have been conducted using a complete apache storm installation in local mode where all services and daemons are installed in one machine as outlined above. Throughout the set of phase-I experiments, the goal is to measure the performance and QoS Metrics through the deployment of the Metrics Collector Module over the various components of the Apache Storm cluster and individual worker nodes.

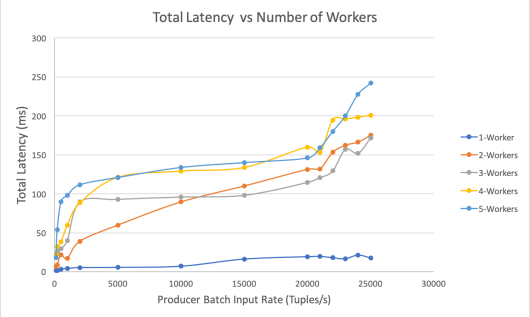

Figure 5. Total Latency of the WordCountTopology running on multiple workers within the same worker node.

1) Storm (In-process) Test Bed: To establish a baseline and investigate the extent of resource utilisation of the experiment test-bed, several throughput tests have been conducted to check how much data the test-bed can consume. Memory-Intensive and CPU-Intensive topologies are being used to generate a rich metrics dataset to be used in the next phase of our project. The experiments were run in local-mode to establish a base line with the setup and configuration of the various components of the Storm platform. The main machine used is a MacBook Pro laptop running Mac OS Catalina version 10.15.7. It is a 2.6 GHz Quad-Core Intel Core-i7 with 16 GB 2133 MHz LPDDR3 memory. The machine run the main Storm daemons (Nimbus, Supervisor and UI) in addition to a zookeeper daemon for coordination and instance management.

A. Preliminary Results and Analysis

1) Metrics Collection: A new metrics system has been introduced in Apache Storm version 2.2. The new system reports the different internal statistics (e.g., acknowledged, failed, emitted, transferred, queue metrics, etc.)

TABLE II
COMPLETE LATENCY FOR NORMAL INPUT

Batch Size	Number of Workers				
	1	2	3	4	5
100	1.67	6.87	22.19	21.72	18.22
200	1.40	8.59	25.70	32.07	54.00
500	3.30	21.54	29.47	38.45	89.62
1000	4.16	16.95	39.85	59.79	98.21
2000	5.20	39.01	89.62	88.45	111.5
5000	5.53	59.79	93.65	121.08	120.9
10000	7.08	89.52	96.36	129.08	133.7
15000	16.2	110.98	98.21	133.72	140.6
20000	19.2	131.27	114.7	159.85	146.0
21000	19.7	131.94	120.9	152.71	159.3
22000	17.8	153.23	129.4	194.67	180.5
23000	16.7	162.24	157.2	196.06	200.1
24000	21.3	166.43	152.1	198.76	227.8
25000	17.6	175	171.6	200.79	242.0

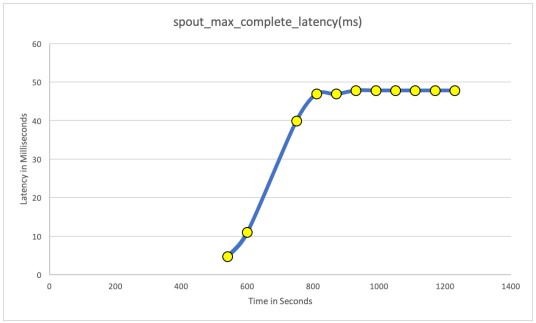

Figure 6. Spout maximum complete latency as reported by the Storm UI.

as well as a new API for user defined metrics. Metrics related to the network-intensive topology are being used in the cluster setup as part of phase II experiments and will be integrated within the main metrics dataset.
During this experiment, the metrics collection process has focused on two levels:

a) Topology Level metrics including the number of worker nodes, workers, memory allocation, cpu allocation, executors, tasks, input data rate, input throughput, emitted tuples, complete latency, maximum overall latency, acknowledged and failed tuples.
b) Component level including the type (Spout/Bolt), executors, tasks, user cpu, system cpu, completed latency, executed tuples, acknowledged and failed tuples, execute latency, operator capacity, process latency, parallelism and congested component.

2) Metrics Measurement: In this set of experiments, topologies that measure the performance of the Apache Storm platform and collect its performance metrics based on the above descriptions and metrics definitions have been utilised. The topologies range from memory-intensive, cpu-intensive and network-intensive topologies.

During the normal operational activities where the available resources are able to handle the fluctuations in the data input rate and the delay/throughput levels are within acceptable

levels of the topology submitter (user). Figures 5, 6 and 7 show the results of running the topology using normal working conditions and high data rates and how such fluctuations affect the system's QoS.

Running the topology using several workers will introduce traffic between inter-workers and intra-workers which affects the performance and latency of the tuples considerably. Such behaviour is evident when comparing the latency using 1 worker and 2 workers. The latency also has a direct relationship with the way the Storm Scheduler places the physical executors and tasks within the same worker. For example, due to the fact that our topology has only three components, placing the executors and tasks of each component of the topology within the same worker will result in lower latency under high input rates with just 3 workers since the inter-executors traffic will be reduced considerably.

When the data input rate exceeds the computing capacities of the allocated resources where the QoS levels degrade dramatically specially the various latency parameters. This is evident in Figures 8 and 9 where the component execution capacity (defined as the number of executed tuples multiplied by the execution latency/(Observation Window Time)) reaches high levels and the buffers start to be full and tuples are dropped or other mechanisms like back-pressure signals are generated to limit the input rates to the topology data ingestion ports (spouts).

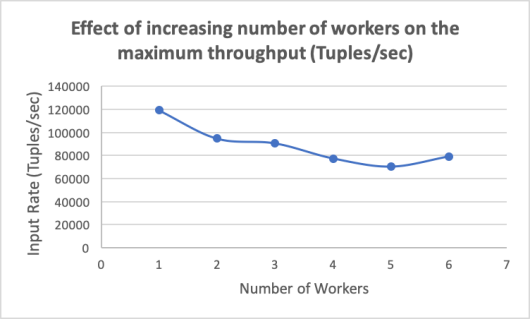

Figure 7. Maximum Input Throughput for the WordCountTopology running on multiple workers within the same worker node.

Maximum Input Throughput for the WordCountTopology running on multiple workers within the same worker node is presented in Table II. During the various runs of our testing topologies, it was observed that the processing power and output rates as well as the various latencies are performing reasonably well when executed in a single worker process within the same worker node. The data input rate to the topology's spout and corresponding worker process was somewhat constrained within a range (under 120k tuples/second).

When the input rate increases, more of the processing power would be used to either (1) drop the tuples instead of processing them or (2) enable the back-pressure techniques recently implemented in the latest versions of Storm to limit the input rate of the incoming data. This can affect the processing rate of the worker and may not be tolerated by certain applications or guaranteed QoS requirements of the application (such as critical infrastructure, military and vital health monitoring systems). A more detailed analysis of the individual worker processes and its components (spouts and bolts) surely will help create better mathematical models and identify bottlenecks and resource starvation or under-use to pinpoint the areas of modifications needed to improve the overall performance of the system.

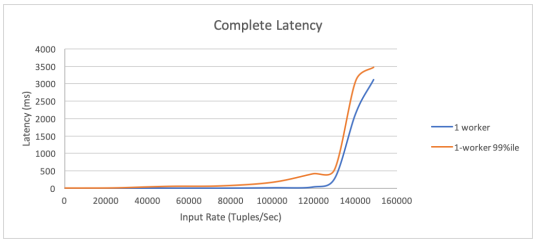

Figure 8. Complete Latency of the Topology as a function of Input Data Rate.

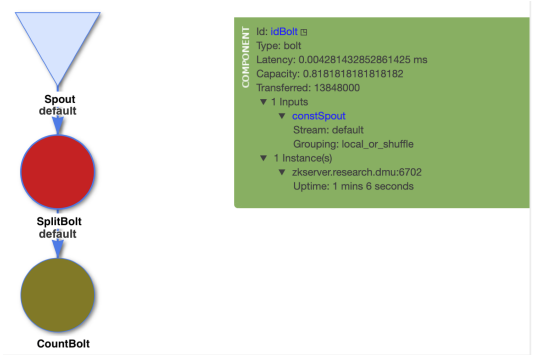

Figure 9. Congestion caused by high input data rates and its effects on the processing capacities of the various topology components.

B. Component Profiling

An improvement to the process is to profile the worker components automatically at runtime. Profiling each individual component (operator) with a topology is time consuming and generally a tedious job that will consume extra "precious" resources from the system and degrade its performance. If the characteristics of the individual component (spout and bolt) changes over time, then it is difficult to depend on the existing processing power, latencies and output throughput to produce suitable resource allocations.

Components can be profiled to monitor the amount of memory, cpu and network bandwidth and then correlate it with the number of tuples being processed in unit time. This will be used to estimate the maximum processing rate the worker can sustain over a time unit so prediction models can be built to anticipate for changes in input rate ahead of time and adjust the resources accordingly.

Figure 9 shows the visualisation of the topology components behaviour when components are congested. Storm deploys a mechanism of back-pressure techniques to limit the input rate in these cases instead of just dropping some of the input tuples.

 ISBN: 978-1-61208-842-6

Spearman's Correlations

			Spearman's rho	p	Lower 95% CI	Upper 95% CI
rate(tuples/s)	-	median(ms)	0.629***	< 0.001	0.587	0.669
rate(tuples/s)	-	cores	0.675***	< 0.001	0.637	0.71
rate(tuples/s)	-	completed	1.000***	< 0.001	1.000	1.000
rate(tuples/s)	-	user_cpu (ms)	0.798***	< 0.001	0.772	0.821
rate(tuples/s)	-	mean(ms)	0.652***	< 0.001	0.612	0.69
rate(tuples/s)	-	ui_complete_latency (ms)	0.730***	< 0.001	0.695	0.762
Median(ms)	-	cores	0.816***	< 0.001	0.792	0.837
Median(ms)	-	completed	0.630***	< 0.001	0.587	0.669
Median(ms)	-	user_cpu (ms)	0.825***	< 0.001	0.802	0.845
Median(ms)	-	mean(ms)	0.987***	< 0.001	0.985	0.989
Median(ms)	-	ui_complete_latency (ms)	0.746***	< 0.001	0.712	0.776
Cores	-	completed	0.675***	< 0.001	0.637	0.711
Cores	-	user_cpu (ms)	0.960***	< 0.001	0.954	0.965
Cores	-	mean(ms)	0.827***	< 0.001	0.804	0.847
Cores	-	ui_complete_latency (ms)	0.890***	< 0.001	0.875	0.904
Completed	-	user_cpu (ms)	0.798***	< 0.001	0.772	0.822
Completed	-	mean(ms)	0.653***	< 0.001	0.612	0.69
Completed	-	ui_complete_latency (ms)	0.730***	< 0.001	0.695	0.762
User_CPU(ms)	-	mean(ms)	0.840***	< 0.001	0.819	0.859
User_CPU(ms)	-	ui_complete_latency (ms)	0.944***	< 0.001	0.935	0.951
mean(ms)	-	ui_complete_latency (ms)	0.768***	< 0.001	0.737	0.796

* p < 0.5, ** p < 0.01, *** p < .001

Figure 10. Correlation Table of partial Metrics based on the Spearmans Correlation Coefficients with %95 confidence intervals.

C. Correlation Analysis

In this section, a statistical techniques called correlation analysis was deployed in order to evaluate the strength of relationship between the various metrics collected from the different components of the topologies and under variable operational environments. High correlation coefficients mean that two or more variables have a strong relationship with each other. Figure 10 shows the correlations based on Pearson's Coefficient with confidence interval of 95.0%. The significant correlations are flagged with (*).

Variable	Input rate (tuples/s)	Median (ms)	Cores	Completed	User_cpu (ms)	Latency (ms)	Mean (ms)
Rate (tuples/s)	1.000	0.629	0.675	1.000	0.798	0.652	0.730
Median(ms)	0.629	1.000	0.816	0.630	0.825	0.987	0.746
Cores	0.675	0.816	1.000	0.675	0.960	0.827	0.890
Completed	1.000	0.630	0.675	1.000	0.798	0.653	0.730
User_cpu (ms)	0.798	0.825	0.960	0.798	1.000	0.840	0.944
Mean (ms)	0.652	0.987	0.827	0.653	0.840	1.000	0.768
Latency (ms)	0.730	0.746	0.890	0.730	0.944	0.768	1.000

Figure 11. Heatmap representation of the correlation between the metrics.

A heat map presentation of the correlation relationships is presented in Figure 11 and highlights the most relevant metrics that will be used in the future experiments.

VII. Conclusions and Future Work

Throughout this paper, a QoS-aware self-adapting resource utilisation framework has been presented with the aim of achieving the following main two goals:

- well-utilisation of system resources (Memory, CPU and network) by continuously predicting resource usage by online machine learning techniques, and dynamically tuning the related parameter configurations of the DSMS,
- reducing tuple response times and maximising system throughput, and satisfying user-specified QoS demand levels of each stream query application.

The rest of the experiments of this research will be carried out using computing instances from Google Cloud Computing Platform. We utilise this platform to simulate the real environment that Apache Storm and other DSMS operates on in order to fully validate the applicability and performance gains of the proposed framework.

References

[1] X. Li, L. Ma, K. Li, K. Wang, and H. A. Wang, "Adaptive Load Management over Real-Time Data Streams," Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007), IEEE, vol. 2, August 2007, pp. 719-725.

[2] Y. Wei, V. Prasad, S. H. Son, and J. A. Stankovic, "Prediction-Based QoS Management for Real-Time Data Stream," In Proceedings of 27th IEEE International Real-Time Systems Symposium, IEEE, December 2006, pp. 344-358.

[3] J. P. Rinne, M. Liljeberg, and J. J. Jouppi, "Quality of service definition for data streams," Nokia Oyj, January 2008. U.S. Patent 7,320,029.

[4] M. D. de Assuncao, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," Journal of Network and Computer Applications, vol. 103, pp.1-17, 2018.

[5] L. Xu, B. Peng, and I. Gupta, "Stela: Enabling stream processing systems to scale-in and scale-out on-demand," In 2016 IEEE International Conference on Cloud Engineering (IC2E), IEEE, April 2016, pp. 22-31.

[6] A. Pagliari, F. Huet, and G. Urvoy-Keller, "NAMB: A Quick and Flexible Stream Processing Application Prototype Generator," In The 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, May 2020, hal-02483008.

[7] A. Muhammad, and M. Aleem, "A3-Storm: topology-, traffic-, and resource-aware storm scheduler for heterogeneous clusters," JOURNAL OF SUPERCOMPUTING, vol. 77, pp. 1059-1093, May 2020.

[8] B. Peng, M. Hosseini, Z. Hong, R. Farivar, and R. Campbell, "R-storm: Resource-aware scheduling in storm," In Proceedings of the 16th Annual Middleware Conference, November 2015, pp. 149-161.

[9] N. Chaudhry, K. Shaw, and M. eds. Abdelguerfi, "Stream data management," Springer Science & Business Media, 2006. vol. 30.

[10] W. Wingerath, N. Ritter, and F. Gessert, "Data Stream Management," In Real-Time and Stream Data Management, Springer, Cham, January 2019, pp. 43-55.

[11] M. Garofalakis, J. Gehrke, and R. Rastogi, "Data stream management: A brave new world," In Data Stream Management, Springer, Berlin, Heidelberg, 2016, pp. 1-9.

[12] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, and S. Khan, "A survey of distributed data stream processing frameworks," IEEE Access, vol. 7, 2019, pp.154300-154316.

[13] "Apache Storm," Apache Software Foundation, 2014. [Online]. Available: https://storm.apache.org/about/integrates.html. [Accessed 01 January 2020].

[14] X. Liu, and R. Buyya, "D-storm: Dynamic resource-efficient scheduling of stream processing applications," In 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS) IEEE, December 2017, pp. 485-492.

[15] Y. Wei, V. Prasad, S. H. Son, and J. A. Stankovic, "Prediction-based QoS management for real-time data streams," In 2006 27th IEEE International Real-Time Systems Symposium (RTSS'06), IEEE, December 2006, pp. 344-358.

[16] R. Gopalakrishnan, and G. M. Parulkar, "A framework for QoS guarantees for multimedia applications within an endsystem," In GISI 95, Springer, Berlin, Heidelberg, pp. 43-56, 1995.

[17] M. HoseinyFarahabady, A. Y. Zomaya, and Z. Tari, "QoS-and contention-aware resource provisioning in a stream processing engine," In 2017 IEEE International Conference on Cluster Computing (CLUSTER), September 2017, pp. 137-146.

[18] N. Tantalaki, S. Souravlas, M. Roumeliotis, and S. Katsavounis, "Pipeline-Based linear scheduling of big data streams in the cloud," IEEE Access, vol. 8, pp.117182-117202, June 2020.

[19] T. Buddhika, R. Stern, K. Lindburg, K. Ericson, and S. Pallickara, "Online scheduling and interference alleviation for low-latency, high-throughput processing of data streams," IEEE Transactions on Parallel and Distributed Systems, vol. 28(12), pp.3553-3569, 2017.

[20] D. J. Abadi et al., "Aurora: a new model and architecture for data stream management," the VLDB Journal, vol. 12(2), pp.120-139, 2003.

[21] A. Arasu et al., "Stream: The stanford data stream management system," In Data Stream Management, Springer, Berlin, Heidelberg, 2016, pp. 317-336.

[22] S. Schmidt, H. Berthold, and W. Lehner, "Qstream: Deterministic querying of data streams," In Proceedings of the Thirtieth international conference on Very large data bases, vol. 30, August 2004, pp. 1365-13.

 ISBN: 978-1-61208-842-6

[23] U. Cetintemel et al., “The aurora and borealis stream processing engines,” In Data Stream Management, Springer, Berlin, Heidelberg, pp. 337-359, 2016.

[24] A. Klein, and W. Lehner, “Representing data quality in sensor data streaming environments,” Journal of Data and Information Quality (JDIQ), vol. 1(2), pp.1-28, Sep. 2009.

[25] H. Wang, Y. Ma, X. Zheng, X. Chen, and L. Guo, “Self-adaptive resource management framework for software services in cloud,” In 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), IEEE, 2019, pp. 1528-1529.

[26] M. A. Serhani, H. T. El-Kassabi, K. Shuaib, A. N. Navaz, B. Benatallah, and A. Beheshti, “Self-adapting cloud services orchestration for fulfilling intensive sensory data-driven IoT workflows,” Future Generation Computer Systems, vol. 108, pp. 583-597, 2020.

[27] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, “Distributed QoS-aware scheduling in Storm,” In Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, Jun. 2015, pp. 344-347.

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: an update,” ACM SIGKDD explorations newsletter, vol. 11(1), pp.10-18, 2009.

[29] A. Bifet et al., “MOA: a real-time analytics open source framework,” In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, Berlin, Heidelberg, Sep. 2011, pp. 617-620.

 ISBN: 978-1-61208-842-6