
TARJANA, YAGNIK,  
Department of Computing, Engineering and Media, De Montfort University, UK  
FENG, CHEN,  
Department of Computing, Engineering and Media, De Montfort University, UK  
LALEH, KASRAIAN,  
Department of Computing, Engineering and Media, De Montfort University, UK

Quality of Service (QoS) has been identified as an important attribute of system performance of Data Stream Management Systems (DSMS). A DSMS should have the ability to allocate physical computing resources between different submitted queries and fulfil QoS specifications in a fair and square manner. System scheduling strategies need to be adjusted dynamically to utilise available physical resources to guarantee the end-to-end quality of service levels. In this paper, we present a proactive method that utilises a multi-level component profiling approach to build prediction models that anticipate several QoS violations and performance degradations. The models are constructed using several incremental machine learning algorithms that are enhanced with ensemble learning and abnormal detection techniques. The approach performs accurate predictions in near real-time with accuracy up to 85% and with abnormal detection techniques, the accuracy reaches 100%. This is a major component within a proposed QoS-Aware Self-Adapting Data Stream Management Framework.

CCS Concepts: • Information systems → Data streaming; Stream management;

Additional Key Words and Phrases: Data Streams, Quality of Service, Prediction Models, Resource Allocation.

ACM Reference Format:  

1 INTRODUCTION

Development of resource allocation strategies and performance prediction using Machine Learning (ML) techniques have been the subject of many research papers [1–4]. Performance prediction using ML algorithms has traditionally been performed over static datasets. The continuous availability of such data sets enables the algorithm to access the data set several times for training and prediction. There is no strict time constraints when manipulating such type of data. On the other hand, data streams have strict processing times associated with them and any analysis, data mining or prediction need to be done on the fly.

Incremental learning algorithms [5] are considered the most suitable machine learning methods to perform prediction of any value specially within the data streams settings. These techniques have the capabilities to offer very high estimation accuracy with minimal computing overhead. They
are able to account for factors such as hardware characteristics of the systems as well as interaction between various components. Such techniques use gradual and step-wise approach to approximate resource usage for each data point as a single input value in terms of the different resource requirements. Incremental learning algorithms use less resources in terms of CPU processing power and memory utilisation. Such algorithms don’t need to store the data and they are usually very fast due to the fact that they need to look into the data elements once to make their analysis or prediction and then the data is totally discarded.

This paper is organised as follows. Section 2 prepare the stage with a brief background on the subject of this research. Section 3 provides an overview of the various components of a Quality of Service management system within a DSMS. Section 4 explores the various incremental learning machine learning algorithms and the various techniques and tools used in building such prediction models. Section 5 gives a glimpse on the current research landscape in the field of QoS of DSMS. Section 6 gives an overview of our proposed framework. Section 7 details our experiments, test-bed configuration along with result’s analysis and discussions. Section 8 concludes the paper and outlines the directions of future work in this research.

2 BACKGROUND

Stream data management has been the subject of many research and industrial activities within the last decade [6]. Several areas of research have been evolved in this field. Each query submission entails the demand to change the configuration parameters based on user’s requirements. This change has a great deal of influence on latency and throughput of the system in addition to resource utilisation. The peculiarity of data stream encompasses a sudden change in arrival rate which stresses dynamic adjustment in parameters configuration to fulfil user’s QoS requirements.

2.1 Data Streams Features and Applications

Data streams can be defined as a sequence of tuples which are naturally generated continuously and infinitely in a specific time interval [7]. The following features distinguish data streams from any other form of data:

- Data streams arrive continuously in real time and possibly for an infinite period of time.
- Streams are coming from different data sources with different data structures.
- Data streams transient so any computations have to be done during this span.
- The data rate or behaviour of data streams can’t be easily predicted.

2.2 Distributed Stream Processing Systems (DSPS)

A DSPS usually has two main layers. The first layer is a Relational Query Systems (RQS) and the second layer is the Distributed Stream Management System (DSMS). Users will submit their applications to the DSPS in the form of a query plan of certain format. The query plan is immediately converted into a directed acyclic graph (DAG) consisting of tasks which are running on the DSPS [8]. DSPS include three main data processing levels and they are:

1. Logical Level: where there are, several inter-connected operators representing the query plans to be run on the data stream. Data streams flow through these operators to undergo different types of computations and data transformations. Such plans are usually presented in Direct Acyclic Graphs (DAGs) and used to form certain configuration plans called Topologies in certain implementation environments such as Apache Storm.
2. Middleware level where DSMS reside such as Apache Storm [9], Yahoo S4 [10], Samza [11] and others.
3. Cloud and cluster-based infrastructures where the servers are configured based on a Master/Slave model.
3 DSMS QOS CONTROL AND MANAGEMENT

A distributed stream processing platform is comprised of multiple layers as follows:

(1) Data Producers: This layer is the entryway to pass the data streams into the distributed processing system.

(2) Ingestion Layer: The collected streams are being pushed and prepared here, then moved to the stream processing system.

(3) Stream Processing Engine: The actual processing will be taken place.

3.1 QoS Specification

A DSMS should enable users to submit and annotate their pre-defined levels of acceptable QoS and associated quality requirements along with the submitted query plans (or Topology Diagrams). The QoS specifications may be annotated in the form of sampling clauses as used by STREAM system [8], quality graphs as used in Aurora system [9], sliding window ranges [10] as used in PIPES or via QoS tables as used in [11].

3.2 QoS Mapping

The QoS mapping mechanism [12] have a clear cost formulas that map any QoS requirements to the needed resource capacity in order to guarantee that those resources are always available and the required QoS level is fulfilled.

3.3 QoS Enforcement

Quality of Service enforcement mechanisms are very important in order to provide a guaranteed QoS level for critical and real-time applications specially in a distributed and shared data streaming and processing system. It is necessary to provide the agreed-upon or contracted service-level Quality of Service (QoS) guarantees to each user in the shared environment.

3.4 QoS Violations

There are two methods to ensure that a DSMS will not violate the QoS requirements of continuous queries over a particular data stream.

- Resources need to be considered carefully in terms of CPU cycles, main memory, network bandwidth and so on, and should be sufficiently allocated in order to support the set of queries over the data stream.
- As soon as the query process is submitted for execution, the DSMS system is expected to experience different types of overload periods due to the fluctuations in the input data rates of the data.

3.5 QoS Metrics

A key feature of our approach is the distinction between three types of Quality of Service (QoS) dimensions related to data streams and their implementation platforms: content-based, query-based, and application-based QoS. Time-based QoS metrics can be defined as the set of indicators about the DSMS’s ability to adapt to different data delivery mechanisms over time and the DSMS speed in processing the different submitted queries. Two main QoS metrics in this category are:

- Throughput: Throughput (or simply the data rate) indicates the amount of stream input which a DSMS is able to handle with its limited amount of resources. Throughput requirements can be specified explicitly by the query owner/user on the bases of a QoS level for each submitted query.
- Latency: Latency is the other main QoS metric which is defined as the time distance from the arrival of an input tuple of the input data stream until this particular tuple has been processed by all the operators contained in a certain query.
4 PREDICTION-BASED RESOURCE ESTIMATION

The issue of resource usage estimation for managing streaming data workload within a DSMS is challenging due to the following distinguished features of data streams:

1. Streamed data arrival rate is not fixed and subject to fluctuations in arrival patterns.
2. The DSMS is processing several types of data and query plans at the same time.
3. Each query plan has its own physical resource requirements and QoS requirements.

4.1 Incremental Learning Algorithms

Incremental learning algorithms [5] are best candidate to perform prediction of any value specially within the data streams settings. The algorithms were selected based on their superiority in handling incremental data sets and their availability within the Weka [13] Incremental Learning toolkit and they are:

1. Bayes Classifiers: Naive BayesGaussian [14] This family of algorithms are based on the Baye’s theorem and utilises one axis-parallel Gaussian distribution per class.
2. Decision Tree Classifiers: Hoeffding Tree (VFDT) [15]is the standard decision tree algorithm for data stream classification. VFDT uses the Hoeffding bound to decide the minimum number of arriving instances to achieve certain level of confidence in splitting the node.
3. Lazy Classifiers: K-Nearest Neighbour (IBk) [16]is a method used for classification and regression. The IBk algorithm generates a prediction model for a test instance just in time.
4. Stochastic Gradient Descent (SGD) [17]: This algorithm learns each task in isolation and from scratch, by fitting a deep neural network over data through extensive, incremental model updates.

4.2 Ensemble Learning

An ensemble of classifiers is a collection of several models combined together. The strength of an ensemble learning approach [18] lies within its ability to gradually and in step-wise fashion by learning from additional ensemble of classifiers. The outputs of those ensembled algorithms are combined through weighted majority voting to obtain the final and most accurate classification algorithm for a particular data set. Such approach improves the accuracy of the predicted models considerably.

4.3 Abnormal Detection

The DSPS cluster nodes are affected by fluctuations that may be caused by the abnormal data rates, spikes or congestions in network transmission and other factors, or the fact that some of the measured metrics and QoS observations don’t fit into the prediction model. Prediction accuracy can be improved by discarding outliers and values of extreme values from the re-trained prediction models. Based on the differences between the observed actual data elements and the predicted ones, the training data is divided into three types:

1. Normal. Where observed data is being predicted correctly by the incremental learning model.
2. Warning. Where observed data is not predicted correctly, but more than half of the nearest data elements are predicted correctly (classified as normal)
3. Outlier. Where the observed data element is way far from the predicted one and more than half of the nearest neighbours are not predicted correctly too.

5 RELATED WORK

A prediction model based on the Mixture Density Network for resource estimation, has been proposed by Khoshkbarforoushha et al. [19]. The model is meant to assist users in making informative decisions about either to submit additional queries to the DSMS or not Yuan Wei et. al. [20]
proposed a Prediction-Based QoS Management for real-time data streams. The model includes certain deadline guarantees for queries submitted to the system based on periodic queries. The work of Q. Jiang and S. Chakravarthy in [21] focused on predicting tuple latency of a query plan. It includes new techniques to enable the DSMS to avoid violating the query plan submitted QoS requirements. Gupta et al. [22] proposed a performance prediction model for big data processing system. The proposal combines information gathered from new active data and historical data utilising machine learning algorithms, Baru et al. [23] stress on the importance of application-level-data benchmarks. Nikravesh et al. [24] introduces an autonomic performance indicator to support scaling in a cloud environment. The major drawback of the proposed models is that most of them are based on static sampling of correlated input and performance datasets.

6 QOS-AWARE SELF-ADAPTING FRAMEWORK

Fig. 1. Qos-Aware Self-Adapting Framework Architecture.

The major work of our research project is the proposal of a QoS-Aware self-adapting resource allocation framework [25]. It enables the collection of dynamic performance and QoS metrics for each component within the DSMS submitted query plan or topology. Figure 1 shows the system architecture of our proposed QoS-Stream DSMS 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.

7 EXPERIMENTAL EVALUATION

Data arrival rates within a DSMS are not persistent and suffer from a great deal of fluctuations over time. Data usually arrives in multiple, continuous, rapid, time-varying streams. One of the major concerns in such environment is the ability of the system to minimise run-time memory usage and response time. In order to evaluate the various components of the proposed framework, several experiments have been conducted using a specifically designated test-bed.

7.1 Test-Bed Setup

Apache Storm Cluster Setup After the completion of the first set of experiments using the local-mode configuration, we have moved to the cluster-based storm configuration. Several computing instances within the Google Cloud Platform were deployed as test bed machines with the specification listed in Table 1.
Table 1. Technical Specifications of the TestBed Machines.

<table>
<thead>
<tr>
<th>No</th>
<th>Machine Name</th>
<th>Ip Address</th>
<th>Memory (GB)</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ZooKeeper</td>
<td>10.154.0.2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Nimbus</td>
<td>10.154.0.7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Kafka</td>
<td>10.154.0.5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Worker1</td>
<td>10.154.0.3</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Worker2</td>
<td>10.154.0.4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>Worker3</td>
<td>10.154.0.6</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

7.2 DSMS Query Plan (Storm Topology)

A Data Stream Processing (DSP) application can be represented by a directed acyclic graph (DAG), where the vertices are the application operators, and the edges are the streams exchanged between operators. For the physical execution of the topology, a DSP is instantiated within the distributed system. This system is usually composed from a set of worker nodes (physical machines), i.e., computational resources or workers, tasks and executors, interconnected together by an overlay network.

7.3 Metrics Collection and Analysis

The metrics reported by Storm provide many insights into how Storm clusters and Storm topologies are performing. Sending these metrics to a metrics module helps in identifying the most relevant and influential set of metrics regarding the specified QoS levels. Such reduced set of metrics is used as an input to a handful of incremental learning algorithms to construct the best prediction models.

7.4 Performance Comparison

The prediction models in this experiment are built based on several incremental learning algorithms and the performance of those algorithms is evaluated. The algorithms take the performance metrics as input and predict a value for either the system complete latency or throughput and compare the values with the user’s specified QoS level of that particular variable to predict if a QoS violation is expected based on the current status of the system.

![Fig. 2. Evaluating the Various Algorithms Mean Errors.](image)

Figure 2 shows three of the main factors which are: the Kappa statistics which takes into account chance agreement within the model along with the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE). Taking those factors into account, it is clear that the lazy-IBK classifier outperforms the rest.

7.4.1 Prediction Correctness. Figure 3 shows the percentage of the correctly predicted values of the different algorithms. It is evident that the Lazy-IBK algorithm outperforms other algorithms
in predicting a correct value of the QoS violations of the whole system. The classifier was able to
correctly predict 1193 data points out of a total of 1473 data points with a correctly classified rate
of 81%. The least performance was from the Bayes-NaiveBayes algorithm with just 64% instances
correctly predicted.

Fig. 3. Comparison of the Various Algorithms Correctly Predicted Rates.

7.4.2 Performance Indicators. Another key performance indicators for the different classifiers are
the classifier’s relative errors. Figure 4 shows that both Lazy-IBK and Function-SGD classifiers are
running head-to-head in this category.

Fig. 4. Prediction ML Algorithms Key Performance Indicators.

7.4.3 Abnormal Detection. In order to improve the accuracy of the algorithms, we looked into
our data points and compared them with the predicted values using the different classifiers. We
analysed the data set to identify outliers and extreme values within the data points. After excluding
those two categories, the correctness of the Lazy-IBK classifier improved to 100%. This proves that
such technique is very powerful in improving the predictability of the model. Improvements were
also observed using other classifiers.

7.5 Component Profiling
In order to identify which executors have the highest impact on the QoS violations, we used the
Storm UI reported component capacity metric since it has the highest effect on other QoS metrics
in the system. We collected performance metrics for each executor within a component and used
the same algorithms to predict the capacity value within a certain range.

Each bolt or spout instance in a topology has a receive queue and a send queue. Each worker
also has a queue for sending messages to other workers. All of these have metrics that are reported.
In this part, we focused our attention on the following two metrics:
The executor capacity is the maximum number of entries in the executor queue. population is the number of entries currently filled in the queue.

(2) Tuple population is the number of tuples currently in the queue as opposed to the number of entries. Those metrics are very important to monitor within Storm and they indicate the current status of the component and help in predicting the direction of its performance.

The \( \text{ExecutorCapacity} \) metric for all executors running as part of a particular component is calculated according to the following equation:

\[
\text{Capacity} = \frac{\text{TuplesExecuted} \times \text{AverageExecutionTime}}{\text{WindowSize}}
\]  

This metric is used to monitor the congestion level of each executor within the topology and as a result the overall capacity of the bolt can be calculated as the average capacity of all executors running within a component. Capacity value that exceeds 50\% results into hiking latency values and degraded throughput, is a vital sign of QoS violation of that particular component and consequently its contribution to the overall QoS violation. Prediction models’ performance comparison for the spout and split components within the topology are shown in Figure 5, and Figure 6 respectively.

### 7.6 Overhead and Computational Costs

In order to evaluate the applicability and usability of the various prediction algorithms within the data stream management system, we looked at the time needed to build the model using each classifier in real-time. Lazy-IBK model outperforms all others in constructing the model within a very negligible amount of time. Figure 7 shows the time taken by each one of the used classifiers.
8 CONCLUSIONS AND FUTURE WORK

In this paper we proposed and implemented a module to predict Quality-of-Service violations based on overall system performance and QoS metrics along with topology component profiling. We investigated the power of incremental learning algorithms and ensemble learning in a challenging real-time data stream environment. The best performed classifier was the Lazy-IBK with accuracy of 81%. Utilising abnormal detection techniques improved the model accuracy to 100%. Applicability and overhead of constructing the prediction models on the overall system resources was also investigated and the IBK classifier proved to be the best.

The results of those experiments will be integrated within the Prediction Module to deploy the module within our main proposed framework. This is a major milestone towards realising our goal in this research project which is to build, verify and validate a QoS-Aware, Self-Adapting Resource Allocation Framework for Data Stream Management Systems.
REFERENCES


