This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2016.2597546, IEEE Transactions on Emerging Topics in Computing

Distribution Based Workload Modelling of Continuous Queries in Clouds

Alireza Khoshkbarforoushha, Rajiv Ranjan, *SMIEEE*, Raj Gaire, Ehsan Abbasnejad, Lizhe Wang, *SMIEEE*, Albert Y. Zomaya, *FIEEE*

Abstract—Resource usage estimation for managing streaming workload in emerging applications domains such as enterprise computing, smart cities, remote healthcare, and astronomy, has emerged as a challenging research problem. Such resource estimation for processing continuous queries over streaming data is challenging due to: (i) uncertain stream arrival patterns, (ii) need to process different mixes of queries, and (iii) varying resource consumption. Existing techniques approximate resource usage for a query as a single point value which may not be sufficient because it is neither expressive enough nor does it capture the aforementioned nature of streaming workload. In this paper, we present a novel approach of using mixture density networks to estimate the whole spectrum of resource usage as probability density functions. We have evaluated our technique using the linear road benchmark and TPC-H in both private and public clouds. The efficiency and applicability of the proposed approach is demonstrated via two novel applications: i) predictable auto-scaling policy setting which highlights the potential of distribution prediction in consistent definition of cloud elasticity rules; and ii) a distribution based admission controller which is able to efficiently admit or reject incoming queries based on probabilistic service level agreements compliance goals.

Index Terms—Data Stream processing workload, Continuous query, Resource usage estimation, Predictable auto-scaling policy, Distribution-based admission controller.

1 Introduction

Efficient resource consumption estimation in response to a query processing task is central to the design and development of various workload management strategies such as dynamic provisioning, workload scheduling, and admission control [6], [27]. All of these strategies typically possess a prediction module which can provide accurate estimations guidance on run-time operations such as adding more resources, reordering query execution, or admitting or rejecting an incoming query.

The data stream processing workload mainly consists of registered continuous queries and data arrival rate distribution models. The key to proper exploitation of elasticity is to have intelligence to predict how changing data velocity and mix of continuous queries will affect the performance of the underlying virtualized resources (e.g. CPU utilization). Therefore, building resource usage estimation for continuous queries is vital, yet challenging due to: (i) variability of the data arrival rates and their distribution models (e.g. logistic); (ii) variable resource consumption of data stream processing workload; (iii) the need to process different mixes of continuous queries; and (iv) uncertainties (e.g. contention) of the underlying cloud resources.

These complexities challenge the task of efficiently processing such streaming workloads on cloud infrastructures where users are charged for every CPU cycle used and every data byte transferred in and out of the datacenter. In this context, cloud service providers have to intelligently balance between various variables including compliance with Service Level Agreements (SLAs) and efficient usage of infrastructure at scales while handling simultaneous peak workloads from many clients.

Recent work has studied SQL query resource estimation and run-time performance prediction using machine learning (ML) techniques [2], [12], [18]. These techniques treat the database system as a black box and try to predict based on the training dataset provided. These techniques offer the promise of superior estimation accuracy, since they are able to account for factors such as hardware characteristics of the systems as well as interaction between various components. All these techniques approximate resource usage for each query as a single point value.

Unlike standard SQL queries that may (not) execute multiple times (often each execution is independent of the previous one), continuous queries are typically registered in stream processing systems for a reasonable amount of time and streams of data flow through the graph of operators over this period. Rapidly time-varying data arrival rates and different query constructs (e.g. time and tuple-based windows) cause the resource demand for a given query to fluctuate over time. To illustrate how streaming workload resource demands fluctuate with time, we executed the following simple CurActiveCars query from a linear road benchmark [5]:

- *A Khoshkbarforoushha is with the Department of Computer Science and Engineering, The Australian National University (ANU) and Data61 CSIRO, Canberra, Australia.*
  E-mail: a.khoshkbarforoushha@anu.edu.au
- *R Ranjan is with the School of Computing Science, Newcastle University, United Kingdom.*
- *R Gaire is with Data61 CSIRO, Australia.*
- *E Abbasnejad is with the Department of Computer Science and Engineering, The Australian National University (ANU) and NICTA, Canberra, Australia.*
- *L. Wang is with Institute of Remote Sensing, Chinese Academy of Sciences, China.*
- *A Y. Zomaya is with the School of Information Technologies, The University of Sydney, Sydney, NSW, Australia.*

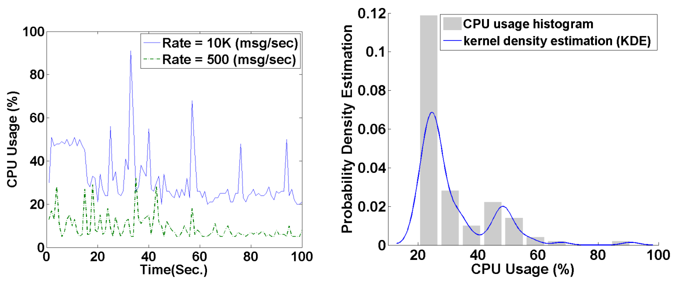


Fig. 1: (a) CPU usage of the query against 500 and $10K$ tuple/sec arrival rates. (b) Normalized histogram and KDE fitted to CPU usage of CurActiveCars query against $10K$ data arrival rate.

```
SELECT DISTINCT car_id
FROM CarSegStr [RANGE 30 SECONDS];
```

Figure 1 (a) illustrates the CPU usage for this query against two different arrival rates: 500 tuple/sec and $10K$ tuple/sec. As expected, the data arrival rates affect the stream processing system resource demand drastically over time. For example, the fitted Probability Density Function (PDF) of the CPU usage for the query (Fig. 1b), shows that even though the query is highly likely to consume between 20% and 35% CPU, we need to allow for possible peak demands (i.e. 90%) to avoid a performance hit. Under these circumstances, how can we address questions such as: *How much memory and CPU share will the query require if the arrival rates double?,What would be the shape of CPU usage for more complex queries?*

For problems involving the prediction of continuous variables (e.g. resource consumption), the single point estimation which is, in fact, a conditional average, provides only a very limited description of the properties of the target variable. This is particularly true for a data stream processing workload in which the mapping to be learned is multi-valued and the average of several correct target values is not necessarily itself a correct value. Therefore, single point resource usage estimation [2], [12], [18] is often not adequate for streaming workload, since it is neither expressive enough nor does it capture the multi-modal nature of the target data. Continuous queries and streaming workload resource management strategies rather require techniques that provide a holistic picture of resource utilization as a probability distribution. To achieve this, we propose a novel approach for resource usage estimation of data stream processing workloads. Our approach is based on the mixture density network (MDN) [7], which approximates the probability distribution over target values.

To illustrate one of the possible advantages of using the proposed approach, consider Figure 2. It displays a sample predicted PDF and actual CPU usage in terms of normalized histogram and fitted Kernel Density Estimation (KDE) for one of the experiments on linear road benchmark [5] queries. As we can see, the estimated PDF approximates the actual resource usage PDF closely. The predicted PDF provides a complete description of the statistical properties of the CPU utilization through which we are not only able to capture the observation point, but also the whole spectrum of the resource usage. In contrast, a best approximation from the existing resource estimation techniques [2], [12], [18] merely provides the point which is visualized by a solid vertical line. Unlike PDFs, with such estimation we are not able to directly calculate any valuable statistical measures (e.g. variance, confidence interval) about the target data.

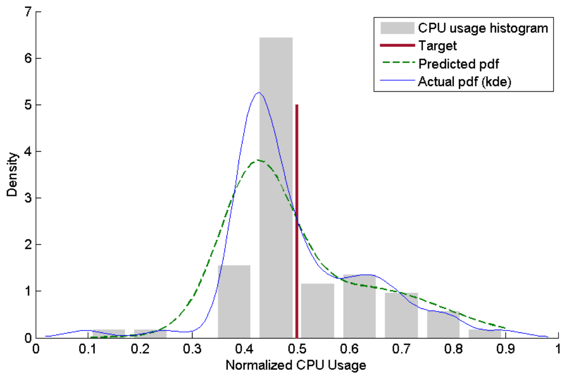


Fig. 2: Sample distribution prediction of CPU usage for NegAccTollStr query. Actual PDF is a fitted KDE function against the actual CPU usage which is used for clarity and comparison with the prediction.

1.1 Summary of Contributions

In summary we make the following contributions:

Distribution-Based Prediction. This paper transfers the solid technique that is MDN from other computer science fields to the cloud and database community. Though other approaches such as Conditional Density Estimation Network and Random Vector Functional Link are also available to estimate the PDF, the benefit of using MDN is its ability to model unknown distributions. In addition, it has already been successfully applied in other domains such as speech synthesis.

Resource Modelling of Continuous Queries. We develop black-box models for predicting CPU and memory usage of centralized data stream processing workloads based on continuous query features and data arrival rates. We also consider resource consumption estimation in the presence of concurrent executions of a large number of queries. Note that the approach makes no assumption of the final shape of distribution which is the key in resource modelling of streaming workload as distribution models can be of any shape and are application specific.

Distribution Based Workload Management. As a concrete demonstration of exploiting the proposed models, we develop two novel applications: i) predictable auto-scaling policy setting; and ii) distribution based admission controller. In the former, we put forward the claim that the workload behaviour distribution prediction provides reliable information enabling consistent auto-scaling policy setting in public clouds. In the latter, we experimentally take the first step towards developing an admission control which is able to react as per the probabilistic SLA. We evaluate our models on a real stream processing system, using both the linear road [5] and TPC-H (www.tpc.org/tpch) benchmarks on both private and public cloud environments.

2 Approach Overview

Figure 3 shows the workflow of our approach as discussed next. In the proposed approach, we use ML technique to train a model on the historical logs. Once the model is built, the workload manager of the stream processing system is able to employ it in order to predict the distribution of a new incoming workload (i.e. query). The predicted PDFs (or mixture models) are then used for different workload management strategies such as admission control, auto-scaling rule setting.

Resource Usage Distribution Prediction: For this purpose, our approach combines the knowledge of continuous query processing with the MDN statistical model. To do so, we firstly execute the training query workload and profile its resource usage values along with predefined query features. Secondly, we input the query features and data arrival rates to the MDN model for training. Following this, the model statistically analyzes the input features' values and actual observation of the resource consumption of the training set and predicts the probability distribution parameters (i.e. mean, variance, and mixing coefficients) over target values. Once the model is built and materialized, it can then be used to estimate the resource usage value of new incoming queries based on the query features' values. Section 4 covers the details of the technique thoroughly.

Auto-scaling Policy Setting: Once the resource distribution prediction becomes available, its exploitation in data stream processing workload management is yet another challenge. Auto-scaling policy setting application demonstrates that the distribution prediction provides a reliable source of information for defining appropriate resource elasticity rules. To do so, the probability of auto-scaling policy activation is calculated. This estimation is then used as a critical parameter for analysing and predicting the impacts of the defined rules on the resources. This feature allows us to define consistent auto-scaling policies or revisit the existing thresholds if needed. More details of the application will be given in Section 5.1.

Distribution based Admission Controller: As another concrete application of the distribution prediction, we develop an admission controller which is able to efficiently admit or reject the incoming queries based on the predicted resource usage PDFs. For this purpose, the SLA miss probability of the incoming workload is calculated. This estimation is then evaluated against different predefined decision making thresholds. This feature enables wise definition of most-to-least probable thresholds simultaneously in order to address different SLA compliances cost-effectively. We will discuss more about the proposed admission controller in Supplementary Section 1.

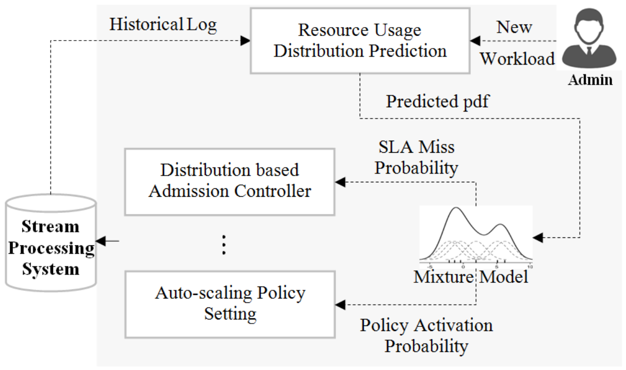

Historical Log
Resource Usage Distribution Prediction
New Workload
Admin
Predicted pdf
SLA Miss Probability
Distribution based Admission Controller
Stream Processing System
Mixture Model
Auto-scaling Policy Setting
Policy Activation Probability

Fig. 3: Our approach builds an MDN model based on the historical logs of queries to pedict distribution of new incoming workloads. The predicted PDFs are then used for developing two novel workload management strategies: a) Distribution based admission control, and b) Auto-scaling policy setting.

3 Related Work

There are two lines of related work; one directly investigates query performance prediction and the other uses estimations for workload management. In this section, we will discuss both and highlight the research gap.

Workload Performance Prediction. Query processing run-time and resource estimation has been investigated in recent years. This line of work explores the estimation of run-time and also resource consumption of SQL queries in the context of both interleaved [2], [10], [18] and parallel execution [1], [10], [21], [24]. In the majority of related work, different statistical ML techniques are applied for query performance estimation. Specifically, techniques such as Kernel Canonical Correlation Analysis (KCCA), Multiple Additive Regression-Trees (MART), and Support Vector Machines (SVM) have been respectively built upon query plan features [12], operator level features [18], or both [2].

When it comes to concurrent workloads, the authors in [1] describe an experimental modelling approach for capturing interactions in query mixes. To do so, the interactions are modelled statistically using different regression models. Along similar lines, [10] argues that buffer access latency measure is highly correlated with the query execution time, and they use linear regression techniques for mapping buffer access latency to the execution times. The authors in [11] also use the k-nearest neighbors prediction technique to identify spoiler model coefficients for the new template based on similar ones. All of the above studies approximate the performance of workload as a single point value. Unlike PDFs, in single point estimates we can not directly obtain valuable statistical measures such as variance or confidence interval about the target data.

Data Processing Workload Management. Workload management and resource sizing for data and stream processing systems use either reactive (e.g. using system load dynamics monitoring) [8], or predictive techniques (e.g. estimating the workload performance) [6], [9], [16], [26], [27] for decision making. In all these predictive approaches, they estimate the workload performance as a single point value [6], [16], assume that the PDF for the workload execution time is already available [27], or estimate (and not predict) the PDF using sampling based techniques [9], [26].

Specifically, [6] proposes an input and query aware partitioning technique which relies on the input rate estimation using time series forecasting. However, predicting workload using a time series analysis is not adequate because event rates usually change in an unpredictable way and a single point estimate does not reflect the distribution. In this regard, although the authors in [27] voiced the issue, they assume that the PDF for the execution time of a query is already available to the service provider. As the single point estimation gives no clue of the confidence on the estimation, they use a committee based ML model in the next work [26]. Along similar lines, [9] approximates the probability distribution using a histogram-based approach. However, this approach is only a simple approximation of distribution based on a number of already collected query execution times. This means it is incapable of predicting the PDF based on the features of a new incoming query.

Concluding Remarks. Based on the above discussions, readers may have noticed the broken link between the two threads of work. Most of the existing techniques for query resource or performance prediction contemplate the target as a single point value, whereas the techniques proposed in recent studies for workload management [9], [26], [27] rely on the whole spectrum of performance or resource usage because even in an OLTP workload, queries with the same query time may follow different query time distributions [27]. The authors in [23] propose a white-box technique for quantifying the uncertainties of query execution time prediction. It treats fixed constant values of operators selectivities, unit cost of single CPU or I/O operation as random variables and develop analytics techniques to infer distribution of likely running times. Although the work differs to ours as they do not target continuous queries and resource usage distribution prediction, it does have the following limitations. The technique is limited to the PostgreSQL optimizer cost model, and more importantly it does not consider concurrent query execution.

Our work attempts to address the above issues by proposing a set of black-box models which are able to predict the distribution of resource usage of highly concurrent workloads. Note that ML algorithms compared to white-box approaches [23], [25] ease the task of cost model generation for increasingly complex data management systems since they are able to capture implicitly the internal behaviour of components and their interaction with OS modules in terms of their resource footprint. This complexity is further intensified in clouds due to the heterogeneity of resource types and uncertainties of the underlying infrastructure.

4 Resource Usage Prediction

4.1 Single Continuous Query

A streaming application is represented by a directed graph whose vertices are operators and whose edges are streams. In our approach, the continuous query feature set and data arrival rate distribution models form the input vector. This exploits an important observation, that data stream processing workload behaviour is predominantly the function of query features along with data arrival rates.

Key to the accuracy of a prediction model is the features used to train the model. We identify a set of potential features that affect the stream processing performance and the query resource usage. The potential features are gathered by analyzing those considered in related work [2], [12] and those we observed in various performance test analyses. Intuitively, not all features have high-correlation with the target of the model and thus we need to select only those features with high predictive capability. To this end, we use a correlation-based feature subset selection method [15] along with best-first search for identifying the most effective attributes from feature vector spaces.

Table 1 lists the feature set used as an input to the model. The attributes are extracted from multiple sources

TABLE 1: Feature input for training model.

Feature Name	Description	Collection Source
avg_arrival_rate	Average arrival rate (tuple/sec)	Distribution Model
stream_no	# of data stream sources	Query statement
subquery_no	# of nested subqueries	Query statement
agg_func_no	# of aggregation functions	Query statement
join_predicate	# of join predicates in query	Query statement
project_size	Projection size of query	Query statement
equ_predicate	# of equality selection predicates	Query statement
inequ_predicate	# of non_equality selection predicates	Query statement
agg_column_no	# of columns involved in GROUP BY clause	Query statement
opt_type_count	# of each operator type in query plan	Query plan
win_type_size	The size of windows which is either time unit (sec) in time window or tuple unit (number) in tuple window type	Query statement
win_type_slide	The sliding value of the window type	Query statement

such as query statement text (e.g. win_type_size), distribution model (e.g. avg_arrival_rate), or query plan (e.g. opt_type_count). Although previous studies [2] showed that the selectivity of operators and cardinality estimates are useful features for execution time prediction, the reason why they were not considered in our feature set is discussed in section 4.2.1. Note that the above list is further customized based on the target prediction, because attributes have different predictive impact on CPU and memory usage estimation. A feature that highly correlates memory consumption might have no correlation with CPU usage. For example, the feature selection task shows that the window size has an insignificant effect on CPU usage prediction, while it affects memory usage prediction heavily.

4.2 Concurrent Workload

A streaming application typically consists of a number of continuous queries simultaneously being processed by the system. This means a resource usage modelling technique has to consider resource consumption estimation in the presence of concurrent executions and the combined workload of a large number of queries.

Queries running concurrently in a mix may either positively or negatively affect each other [1]. Therefore, to model a concurrent workload, we need to study: i) the way a system runs a batch of queries and applies optimizations to reflect possible positive interaction in the feature set, and ii) the way queries compete for shared hardware resources to identify possible negative effects on the mix performance. These two issues are studied in the following sub-subsections respectively.

4.2.1 Stream Processing Optimizations

The first step toward modelling concurrent workload is feature set extension. This process is, in fact, adapting the features for isolated query resource usage prediction to include features from concurrent executions. Since the proposed technique is based upon continuous query features, the key to successful modelling of combined workloads is the function of understanding the way the system applies optimizations. The main optimization techniques are operator reordering, redundancy elimination, placement, state sharing, and so on [17] that are somewhat supported by today's stream processing systems. For example, Odysseus supports query rewrite (e.g. selection and projection push down) and query sharing. Note that the mentioned optimization strategies are not exclusive to multi-query execution. However, some strategies such as sub-graph sharing or state sharing are more likely to be applied in the case of concurrent workload.

According to the initial feature set selection (Table 1), three optimization strategies including redundancy elimination, state sharing, and reordering need to be investigated for feature set extension. Because the others are either i) not applicable due to the scope of this study (e.g. operator placement which is for distributed stream processing environment), ii) application specific (e.g. load shedding that trades performance against accuracy of results), or iii) related to system performance configuration (e.g. batching which is a typical performance tuning option in stream processing systems such as Oracle CEP).

Redundancy Elimination. In case of multiple-query registration, a data stream processing system constructs a global query graph, which contains all operators of all currently active queries in the system. In this case, a query optimization component is used to detect reusable operators in different queries. For example, the Odysseus stream processing system [3] applies query sharing which uses one operator in case of existing multiple same operators in different queries from the sources to the sinks. Therefore, for concurrent workload we include a list of *distinct* query execution plan nodes (i.e. operators) for all the queries in our training set as opposed to a single continuous query. This defines a global feature space to describe all concurrent queries in the workload.

State Sharing. This strategy optimizes for space by avoiding unnecessary copies of data. For example, continuous query language (CQL) implements windows by non-shared array of pointers to shared data items, such that a single data item might be pointed to from multiple windows [4]. Therefore, when there are multiple window operators against the same source, we consider the largest window size (i.e. win_type_size) in the feature list.

Operator Reordering. Reordering is profitable when there is a chance to move selective operators before costly ones. For example, Odysseus [3] applies selection and projection push-down which avoids unnecessary processing. This optimization which is typically performed by the optimizer affects the selectivity ratio (i.e. the number of output data items per input data items) of the operator even in single query execution. However we did not include the selectivity ratio of the operator as a feature to our training vector since in a stream processing environment we do not have *control* over the selectivity of the operators due to consistent data arrival rate fluctuations. Moreover, preliminary investigation of the influence of operator selectivity using a sampling approach in a set of experiments found that its contribution to the accuracy of *resource usage* distribution prediction is negligible.

4.2.2 Resource Contention

When multiple queries are registered on the same host, the operators competing for common hardware resources such as disk, memory, or CPU might negatively impact performance. As we aim at resource usage modelling, the resource contention is not a challenge because our models capture the overall resource utilization. This means if there is a contention we will encounter higher CPU utilization and vice versa. Thus, the contention issue is implicitly handled by our models.

Resource contention hits query performance such as latency and throughput. Although prediction of these measures is not in the focus of this study, our approach to resource usage modelling paves the way for scrutinizing the concurrency impact on query performance prediction. Specifically, distribution based prediction of resource utilization for a given query when it runs either in isolation or in a mix provides upper and lower bounds of resource usage. Based on this information, analytical or statistical models (e.g. correlation) are able to describe how query performance varies under different resource availability scenarios.

4.3 Model Selection

The classic statistical ML techniques such as multilayer perceptron (MLP) are able to model the statistical properties of data generator. However, if the data has a complex structure, for example it is a one-to-many mapping, then these techniques are inadequate [22]. The scatter plot of CPU usage against average arrival rates in CurActiveCars query (Figure 4) illustrates the multi-valued mapping point in which for the same arrival rate such as $10K$ (tuple/sec) there are many CPU usage values which range from 20 to 90 percent. This means that the conditional distribution for many input value such as $10K$ or 9998 is multi-modal. Such a multi-modality can be poorly represented by the conditional average. Therefore, we need a technique that is able to capture the multi-modal nature of the target data density function. Note that such behavior in data stream processing workloads is common because the window construct has the potential to impose such significant variations in resource demands even if we disregard arrival rate fluctuations or performance violation from other workloads.

Our approach employs MDN [7], a special type of Artificial Neural Network (ANN), in which the target (e.g. CPU usage) is represented as a conditional PDF. The conditional distribution represents a complete description of data generation. An MDN fuses a mixture model with an ANN. We utilize a Gaussian Mixture Model (GMM) based MDN where the conditional density functions are represented by a weighted mixture of Gaussians. The GMM is a very powerful way of modelling densities, since it is able to fully describe models by three parameters that determine Gaussians and their membership weights. From this density, we can calculate the mean which is the conditional average of the target data. Moreover, full densities are also used to

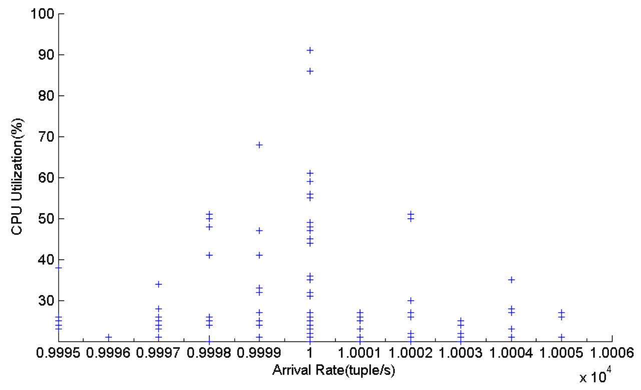


Fig. 4: CPU usage of CurActiveCars query against average arrival rates showing the multi-valued mapping situation from the same input.

accurately estimate expectation and variance that are two main statistics characterizing the distribution.

Figure 5 gives an overview of the approach. The main input features of the model consists of collected query features from the CQL statement and query plan. In this process, the neural network is responsible for mapping the input vector x to the parameters of the mixture model $(\alpha_i, \mu_i, \sigma^2)$, which in return provides the conditional distribution. In fact, Figure 5 shows a sketchy example MDN with 2 components that takes a feature set x of dimensionality 4 as input the vector and provides the conditional density $p(t|x)$ over target t of dimensionality 1.

A number of other approaches such as Conditional Density Estimation Network and Random Vector Functional Link are also available to estimate the PDF. The benefit of using MDN is due to its ability to model unknown distributions as exhibited by continuous queries and streaming workload.

4.3.1 Mixture Density Networks

The combined structure of feed-forward neural network and a mixture model make an MDN. In MDN, the distribution of the outputs t is described by a parametric model. The parameters of this model are determined by the output of a neural network. Specifically, an MDN maps a set of input features x to the parameters of a GMM including mixture weights α_i, mean μ_i, and variance σ^2 which in turn produces the full PDF of an output feature t, conditioned on the input vector $p(t|x)$. Thus, the conditional density function takes the form of GMM as follows:

$$p(t|x) = \sum_{i=1}^{M} \alpha_i(x)\phi_i(t|x) \tag{1}$$

where M is the number of mixture components, ϕ_i is the ith Gaussian component's contribution to the conditional density of the target vector t as follows:

$$\phi_i(t|x) = \frac{1}{(2\pi)^{c/2}\sigma_i(x)^c} exp\left\{-\frac{||t-\mu_i(x)||^2}{2\sigma_i(x)^2}\right\} \tag{2}$$

The MDN approximates the GMM as:

$$\alpha_i = \frac{exp(z_i^{\alpha})}{\sum_{j=1}^{M} exp(z_j^{\alpha})} \tag{3}$$

$$\sigma_i = exp(z_i^{\sigma}) \tag{4}$$

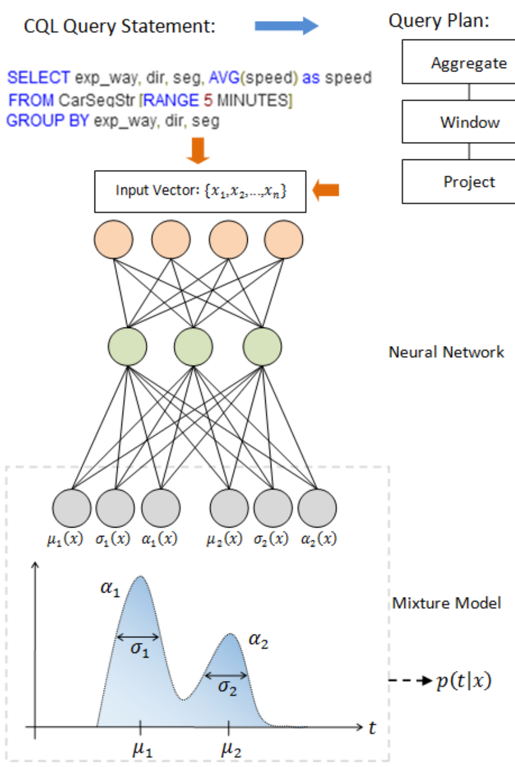

CQL Query Statement:
Query Plan:
SELECT exp_way, dir, seg, AVG(speed) as speed
FROM CarSeqStr [RANGE 5 MINUTES]
GROUP BY exp_way, dir, seg
Aggregate
Window
Project
Input Vector: $\{x_1, x_2, \ldots, x_n\}$
Neural Network
$\mu_1(x)$ $\sigma_1(x)$ $\alpha_1(x)$ $\mu_2(x)$ $\sigma_2(x)$ $\alpha_2(x)$
Mixture Model
α_1
σ_1
α_2
σ_2
μ_1
μ_2
t
$p(t|x)$

Fig. 5: Overview of the proposed approach for predicting the resource usage distribution of continuous queries.

$$\mu_i = z_i^{\mu} \tag{5}$$

where z_i^{α}, z_i^{σ}, and z_i^{μ} are the outputs of the neural network corresponding to the mixture weights, variance, and mean for the ith Gaussian component in the GMM, given x [7]. To constrain the mixture weights to be positive and sum to unity, the *softmax* function is used in Eq. (3) which relates the output of corresponding units in the neural network to the mixing coefficients. Likewise, the variance parameters (Eq. 4) are related to the outputs of ANN which constrains the standard deviations to be positive.

The objective function for training the MDN is to minimize the Negative Log Likelihood (NLL) of observed target data points given to mixture model parameters:

$$E = -\sum_{n} ln\left\{\sum_{i=1}^{M} \alpha_i(x^n)\phi_i(t^n|x^n)\right\} \tag{6}$$

Since the ANN part of the MDN provides the mixture model parameters, the NLL must be minimized with respect to the network weights. To minimize the error function, the derivatives of the error function with respect to the network weights are calculated. Specifically, the derivatives of the error are calculated at each network output units including the priors, means and variances of the mixture model and then propagated back through the network to find the derivatives of the error with respect to the network weights. Therefore, non-linear optimization algorithms such as scaled conjugate gradients can be applied to MDN training.

5 Distribution-Based Workload Management

Before presenting the experimental results of the proposed technique, here we discuss its applications to answer the following key question: Is the proposed approach applicable to resource management problems of stream processing systems in practice?

5.1 Predictable Auto-Scaling Policy Setting

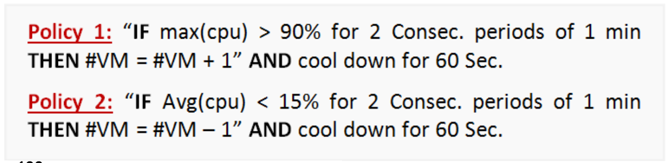
Developing efficient and stable auto-scaling techniques in cloud environments is a challenging task due to heterogeneous infrastructure and transient behaviour of workloads. A number of studies approach this problem with the aid of control theory [19], reinforcement learning [16], and the like. The hard challenge is to determine a suitable *policy* for the decision maker (e.g. resource provisioner), as poor policy settings can lead to either resource inefficiency or instability. For example, consider CPU utilization of the NegAccTollStr and its corresponding auto-scaling policies, as shown in Figure 6 (a). Note that these two policies are defined to avoid SLA misses [1] and resource dissipation respectively.

For the NegAccTollStr query, as the peaks go beyond 90% within 2 consecutive periods of 1 minute, the first policy is triggered and an additional virtual server is instantiated to process the workload (e.g. via stream redirectory technique). However, the load may now drop far below the predefined threshold of the second policy (i.e. avg(cpu)<15%) as the combined capacity of two virtual servers exceeds the current stream processing demands. Therefore, the second policy is activated and the provisioner decreases the number of instances to one. This oscillatory behaviour can continue indefinitely depending on the variation in data stream arrival rate and continuous query processing resource consumption pattern. [19] also reports the same observations. To avoid oscillations, [19] develops the proportional thresholding technique which in fact works by dynamically configuring the range for the controller variables. Though this technique can tackle the oscillatory problem at run-time, it is incapable of anticipating the effects of auto-scaling policies before workload execution which can lead to SLA violations.

To circumvent the limitation of existing approaches, we propose a novel approach as discussed next. We perceive that the reason for the oscillations is due to defining inconsistent policies that are agnostic to changes in workload behaviour. In our approach such inconsistencies are avoided by exploiting the workload distribution prediction for specifying and selecting auto-scaling policies. For example consider SegToll resource usage behaviour as shown in Figure 6 (b) in which only the first policy will be triggered. Based on the workload distribution we do not expect to meet the second policy and following instability even after initial resource resizing.

Based on this observation, we claim that the workload behaviour distribution prediction provides more reliable advice for auto-scaling policy setting. In fact, having the understanding about the upper and lower bound of resource utilization helps in anticipating auto-scaling policy effects beforehand and adjust the configurations accordingly. In other words, a workload-distribution driven auto-scaling policy setting approach can help administrators in defining more consistent auto-scaling policies.

1. We assume that CPU utilization above 90% leads to SLA misses. We will discuss about this relationship in Supplementary Section 1.


Policy 1: "**IF** max(cpu) > 90% for 2 Consec. periods of 1 min **THEN** #VM = #VM + 1" **AND** cool down for 60 Sec.

Policy 2: "**IF** Avg(cpu) < 15% for 2 Consec. periods of 1 min **THEN** #VM = #VM – 1" **AND** cool down for 60 Sec.

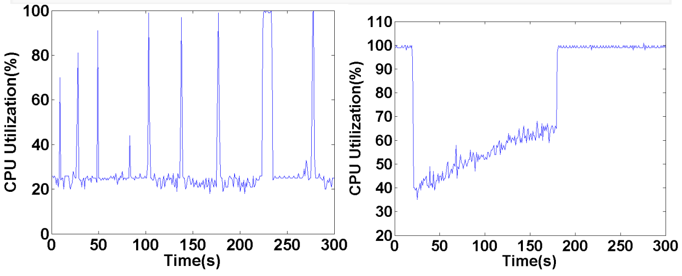


Fig. 6: The CPU utilization of (a) NegAccTollStr and (b) SegToll queries for 5 minutes. The sample auto-scaling policies cause osiliation behaviour in NegAccTollStr workload, since they have been defined irrespective of the workload CPU usage distribution.

To validate the hypothesis, we designed an experiment to evaluate whether the predicted distribution is able to *characterize the most/least probable auto-scaling policies before the actual workload execution or not.*

Workload: A representative workload was built based on the Linear Road Benchmark (LRB), LRB Mix_EC2. The workload contains 5507 execution traces for 17 query mixes. The mixes are at multiprogramming level (MPL) range from 2 to 5. All the mixes were logged for about 4 minutes on Amazon t2.micro instance. To make the test workload, we randomly selected 32 mixes of queries – different from the training set – at MPL range from 2 to 5.

Auto-scaling Policy Generation: In the next step, 128 random auto-scaling policies were generated. The 128 polices were randomly split into 32 sets, each corresponds to a test query mix. This means each mix (out of 32 mixes) is run against a group of 4 auto-scaling policies. Therefore, before running each of the query mixes, 4 auto-scaling policies are defined on t2.micro EC2 instance. We developed all the policies as per the Amazon EC2 template:
Policy Template: Take action A[2] whenever {Average, Max, Min} of CPU Utilization is $\{>, \geq, <, \leq\}$ than γ for at least {2, 5} consecutive periods of {1, 5} minutes.
where the threshold γ was randomly generated in the range (0,100) percent.

Training the Model: We trained the MDN classifier based on the LRB Mix_EC2 training set. We then used the trained model to predict the PDFs of CPU usage for the query mixes of test dataset. In the next step, the probabilities of the policies were calculated based on the predicted PDFs before any workload execution takes place. Once probabilities were calculated, all the query mixes were run one after another against the predefined rules on the EC2 instance and all the activated policies were recorded over the experiment period. The experiment duration was specified according to the policy monitoring period. In our experiment it was twice the monitoring duration[3]. This workflow was continued for all 32 mixes in test dataset.

2. In our experiment it is a simple notification email.
3. For example, the experiment duration is 4 minutes for a policy with a monitoring duration of 2 consecutive periods of 1 minute.

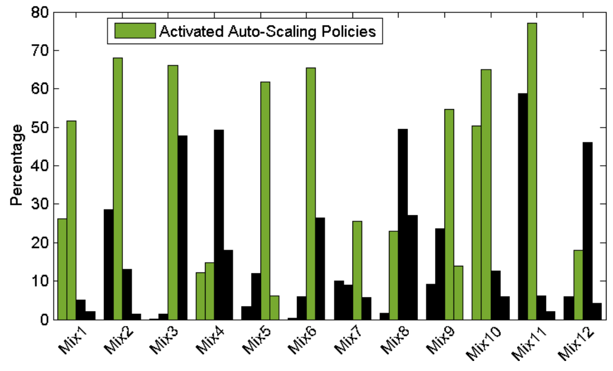


Fig. 7: The probabilities of the randomly generated auto-scaling policies for 12 (out of 32) mixes of test queries. Each query mix evaluated against 4 auto-scaling policies as shown in the form of bright and dark coloured bars. The bright and dark bars within each policy set respectively show the *activated* and *not activated* rules at run-time. Our technique has successfully characterized the highly possible policies for all mixes but Mix 4, 8, and 12.

We now discuss how to calculate the auto-scaling policy probability. To do so, we first compute the probability of the CPU utilization using the following equation:

$$Pr\left[a \le X \le b\right] = \int_a^b f_X(x)dx \tag{7}$$

Where a random variable X has density f_X and the variables a and b are the CPU utilization thresholds. Eq. 7 gives the probability of CPU (or memory) utilization within the given thresholds. However, the auto-scaling policies are also dependent on the consecutive occurrence of the events (i.e. the condition). The events that lead to activation of thresholds are independent of time. Based on the probability theory, assuming independence (i.e., the probability of an event such as a threshold activation at a given point of time is independent from the past occurrence of the same type of event), we can compute the probability of two or more independent events by multiplying their individual probabilities. Therefore, the probability of an auto-scaling policy occurrence for m consecutive periods is calculated in a general form as:

$$Probability(policy, m) = \prod_{k=1}^{m} Pr_k\left[a_k \le X \le b_k\right] \tag{8}$$

The above definition relaxes the constraint of having the same thresholds for arbitrary consecutive periods, though existing auto-scaling frameworks (e.g. Amazon Auto-Scaling Service, Azure Fabric Controller) do not offer this important feature yet. We note that the PDFs do not reflect the probability of workload behaviour across time. However, we show in our experiment that extending the probabilities to an arbitrary number of consecutive periods works well in practice.

Before discussing the results, let us recap the purpose of the experiment. There are 32 rule groups corresponding to 32 query mixes. Each group contains 4 auto-scaling policies of which two are the most and least probable policies as regards to the calculated probabilities. This means they are highly likely and highly unlikely to be triggered after workload execution. We now want to evaluate, for example, *What percentage of the rules with the highest probability values are activated?*

Based on the experimental results, we found that 62% of rules with the highest probability were activated after workload execution. Moreover, 87% of rules that were characterized as unlikely to be triggered at run-time also held true (i.e. they were not triggered). Figure 7 displays the results for 12 out of 32 test query mixes. As the bar chart shows, the proposed technique performs well in predicting the most probable auto-scaling policies for each policy group. As we can see, it only failed to characterize the highly possible policies for Mixes 4, 8, and 12.

In summary, these findings clearly demonstrate that our hypothesis held true and the distribution-based prediction provides a reliable source of information for predictable auto-scaling policy setting. Apart from its contribution to oscillatory behaviour avoidance, we believe that this feature helps users to use cloud infrastructure economically where they are charged for every CPU cycle used and every data byte transferred in and out of the datacenter.

6 Experiment

In this section we evaluate the performance of the approach as regards to the state of the art single point prediction techniques. We conduct our experiment on both public and private clouds to evaluate the accuracy of estimations in the presence of any possible performance variations. However, as we obtained identical results from the experiment on Amazon public cloud, those results are omitted.

6.1 Experimental Setup

Two virtual machine (VM) instances, one for load generation and another as a host for the stream processing system, were employed from CSIRO private cloud. The stream generator system was a m1.medium size instance with 4GB RAM, 2 VCPU running Ubuntu 12.04.02 Server 64b. All queries were executed on m1.large instance size with 8GB RAM, 4VCPU, and the same OS. The hypervisor is KVM, and the nodes are connected with 10GB Ethernet.

6.1.1 Dataset and Workload

To validate our approach we deployed both linear road benchmark (LRB) [5] and slightly modified TPC-H in a commercial centralized stream processing system X.

LRB Workload. This workload has primarily been designed for comparing performance characteristics of streaming systems. It contains 20 queries with different levels of complexity in terms of execution plan. We treated them as template queries. Excluding the ad-hoc query answering set reduced them to 17 template queries. Various arrival rates (e.g. from 100 to 100k tuple/second) along with random substitution of window size (e.g. from 1 to 900 sec.) resulted in 17289 execution traces. To generate data streams, 500MB data (i.e. 3 hours simulated traffic management data) was fed into the streaming system using the system's built-in load generator which played the role of data driver in the LRB. Each query was registered and logged for more than 3× of its window size to capture the impacts of time windows on resource consumption properly.

LRB Mix Workload. To build a representative workload of concurrent query executions, we collected 585 execution traces for 18 query mixes. To generate the dataset, different combinations of the queries at multiprogramming level (MPL) range from 2 to 17 were randomly selected and registered in the stream processing system. Once the mixes start processing of the incoming data streams the CPU and memory usage of the system are collected.

TPC-H Workload. In contrast to the LRB workload, TPC-H has been designed primarily for DBMSs, though it has also been used in stream processing research. In this context, each relation is considered as a data stream source and the tuples are sent toward the stream processing engine over the network using a load generator. Therefore, each registered query references a subset of the relations in the input over time.

We created 0.1GB TPC-H database using the DBGen tool as per the specification. To keep the overall experimentation duration under control we did not use larger database size (e.g. 1GB) because the tables are in fact the stream source material in our experiment and we have to send each tuple over the network. Quite simply, in 1GB database size, LINEITEM table has 6001215 tuples and even with the 5000 tuple/sec rate, it takes more than 20 minutes to send all the tuples over the network. With current hardware, this rate is the maximum consumption rate for queries without any join such as Q1 and Q6. This rate drops to less than 200 for Q8 with 7 data stream sources. As the system X does not support correlated sub-queries, we were forced to exclude templates 4, 11, 15-18, 20-22. We generated 35 executable query texts using QGen based on the remained 13 TPC-H templates queries.

Furthermore, we slightly modified these queries for our system to make them compatible with the stream processing context. One of the key changes was adding a time window for each stream source to let queries show the upper bound of CPU and memory usage. Moreover, some query semantics require that tuples not to leave the time window until a certain period of time to be able to produce meaningful results. In other words, we needed to keep the first tuple that enters the time window until the load generator reads and sends the last tuple from the relation source. To this end, we set the time window range to the value of *S* if the load generator needs *S* seconds to read and send all the tuples.

The load generator was not allowed to send duplicate tuples. In addition, relations have different cardinalities so that in case of multiple stream sources in one query, we set all the time windows to the biggest one. This let the relation at time *t* consist of tuples obtained from all elements of stream up to *t*. For example, in a join between LINEITEM ($\sim$600k tuples) and NATION (25 tuples) streams, the latter requires as big a time window as the former to let elements remain in the window until the last tuple from the LINEITEM stream enters the window for processing.

The 35 generated executable queries were registered separately in the system X and their performance measures against the fluctuating arrival rates were logged. The obtained workload consists of 8783 execution traces.

Performance measures of interests (i.e. CPU and memory) were collected using the *dstat*[4]. This is a lightweight Python-based tool that collects OS and system statistics passively, without affecting performance. To guarantee healthy and repeatable data gathering, the execution traces of all queries were collected several times. Moreover, all queries were run with cold start making sure the buffers were flushed and we had a fresh JVM.

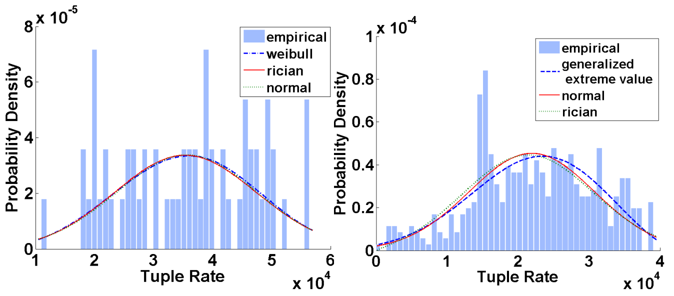


Fig. 8: Best fit of sent tuple per second against different distribution models. The figures contain probability density of average tuple sent per second for the speed rate of (a) $50K$ and (b)$100K$ for two different queries.

Note that all the models are trained and tested as per varying data arrival rate distribution models. To this end, after setting a certain data arrival rate, the generator typically tries to reach the specified velocity, while adjusting rate based on engine consumption rate with the aid of a thread sleep function. This means that a query (especially complex ones) might be able to consume only 100 tuples per second even when we set the load generation rate to 200 tuples per second. Thus, a few seconds after commencement the buffer of the stream processing engine is full and the load generator thread sleeps for a few milliseconds to allow the consumer to exhaust the queue. This situation inherently emulates different load generation distribution, for example for rate $50K$ and $100K$ the distribution is more fit to Weibull and generalized extreme value distribution, as shown in Figure 8.

6.1.2 Training and Testing Settings

To assess how the result of a predictive model would be generalized to an independent unforeseen data set, we divided the LRB workload randomly into training and testing datasets with 66% and 34% split rates respectively. For TPC-H workload we used *k-fold cross-validation*. As regards to the workload size, *2-fold cross-validation* was used to train and test parameters. For each fold, we randomly assigned data points to two equal size sets *ds1* and *ds2*. To do so, we shuffled the data array and then divided it in to two arrays. We then trained on *ds1* and tested on *ds2*, followed by training on *ds2* and testing on *ds1*.

Before training and testing, the input and output features were normalized using z-score and min-max normalization with range (0.1-0.9). For conducting training and testing, we used a Netlab toolbox [22] which is designed to provide the central tools necessary for the simulation of theoretically well founded neural network algorithms and related models and in particular MDN. The implemented MDN model uses a MLP as a feed forward neural network, though in general any non-linear regressor can be utilized.

4. http://dag.wiee.rs/home-made/dstat/

There are a number of hyper-parameters including the number of Gaussian components or number of neurons in MLP that need to be specified beforehand. We evaluated several settings and assessed the trade-off between accuracy, training time and overhead. We concluded that a GMM with 3 components and 2 neurons per feature in the input vector provide an acceptable accuracy within a tolerable overhead.

State of the Art Techniques. To compare the performance of our approach with single point estimators, we used REPTree and SVM as the alternative techniques. REPTree and SVM are the main prediction techniques used in [27] and [2] respectively. Note that [18] also uses a variant of regression trees as a core predictor.

6.2 Evaluation: CPU and Memory Usage

To determine whether a probabilistic model performs well we must set the goal of the model because if, for example, a trained MDN assigns some probability to the actual observation, we should be able judge whether the prediction is accurate or not. Therefore, in the following subsection we first set the goals and then define appropriate metrics.

6.2.1 Error Metrics

The goal of a probabilistic prediction is to maximize the sharpness of the predictive distributions subject to calibration [13]. Sharpness refers to the concentration of the predictive distributions. Calibration refers to the statistical consistency between the PDFs. Our objective is to predict calibrated PDFs that closely approximate the region in which the target lies with proper sharpness. To this end, the Continuous Ranked Probability Score (CRPS) [13] is a proper metric to evaluate the accuracy of PDFs:

$$CRPS(F,t) = \int_{-\infty}^{\infty} \left[F(x) - O(x,t)\right]^2 dx \qquad (9)$$

where F and O are the Cumulative Distribution Function (CDF) of prediction and observation distributions respectively. $O(x,t)$ is a step function that attains the value of 1 if $x \geq t$ and the value of 0 otherwise.

To calculate CRPS both the prediction and the observation are converted to CDF. The CRPS compares the difference between CDF of prediction and observation as given by the hatched area in Figure 9. It can be seen that the area gets smaller if the prediction distribution concentrates probability mass near the observation, i.e. the better it approximates the step function. Moreover, the small CRPS value shows that the prediction captures the sharpness of prediction accurately. After calculating the CRPS for each prediction, we need to average the values to evaluate the complete input set:

$$CRPS = \frac{1}{n}\sum_{i=1}^{n} CRPS(F_i, t_i) \qquad (10)$$

We are also interested in evaluating the spread of predictive density in which our targets lie. The average Negative Log Predictive Density (NLPD) [14] error metric is used for evaluating this aspect:

$$NLPD = \frac{1}{n}\sum_{i=1}^{n} -log(p(t_i|x_i)) \qquad (11)$$

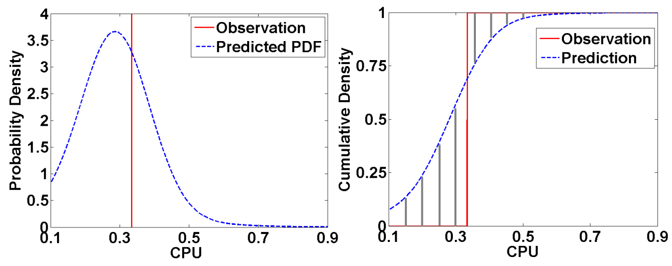


Fig. 9: (a) predicted PDF and the observation (b) schematic sketch of the CRPS as the difference between CDFs of prediction and observation.

where n is the number of observations. The NLPD evaluates the amount of probability that the model assigns to targets and penalizes both over and under-confident predictions.

The last metric is the Mean-Square Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(t_i - m_i)^2 \qquad (12)$$

where m refers to the median of the PDFs as point predictions for the MDNs. This metric allows us to compare the proposed technique with single point competitors.

6.2.2 Evaluation Results

The results for both the proposed approach using MDN and the single point estimators under CRPS, NLPD, and MSE metrics are shown in Tables 2 to 4 respectively. Note that different MDN architectures including 3, 5, and 8 mixture components (M) were evaluated to analyse the influence of this hyper-parameter in the model.

TABLE 2: Trained classifiers performance as per LRB workload.

MDN					**REPtree**	**SVM**
Res.	M	CRPS	NLPD	MSE	MSE	MSE
CPU	3	**0.036**	**-1.95**	**0.006**	0.008	0.007
	5	0.128	-0.339	0.096		
	8	0.113	-0.865	0.043		
Mem.	3	**0.042**	**-3.136**	**0.010**	0.008	0.015
	5	0.053	-1.465	0.066		
	8	0.065	0.075	0.046		

TABLE 3: Trained classifiers performance as per LRB Mix Workload.

MDN					**REPtree**	**SVM**
Res.	M	CRPS	NLPD	MSE	MSE	MSE
CPU	3	0.114	**-0.584**	**0.032**	0.038	0.013
	5	0.106	-0.544	0.085		
	8	**0.099**	-0.46	0.056		
Mem.	3	0.081	**-1.96**	**0.010**	0.011	0.02
	5	**0.042**	-1.33	0.058		
	8	0.068	-1.18	0.042		

TABLE 4: Trained classifiers performance as per TPC-H workload.

MDN					**REPtree**	**SVM**
Res.	M	CRPS	NLPD	MSE	MSE	MSE
CPU	3	**0.034**	**-2.04**	**0.007**	0.006	0.008
	5	0.16	-0.98	0.02		
	8	0.154	-0.9	0.02		
Mem.	3	**0.057**	**-1.9**	**0.008**	0.006	0.011
	5	0.092	-0.91	0.094		
	8	0.097	-0.67	0.1		

All three metrics are negatively oriented scores; hence smaller value is better. Let us first evaluate the accuracy of the MDN per se using CRPS and NLPD measures. As we can see, in three workloads the error numbers are small enough to suggest that the proposed model is an appropriate one for distribution prediction of data stream processing workloads. In LRB Mix workload, sophisticated MDN architecture with 8 and 5 components led to respectively better CPU and memory utilization prediction under CRPS metric. In contrast, both LRB and TPC-H workloads show slightly worse performance as the architecture becomes more complex.

The MDN shows slightly better performance in memory utilization prediction of LRB compared with TPC-H in terms of CRPS values, though its performance in CPU prediction in both workloads is nearly identical. This is why the TPC-H workload is more complex than LRB as the query templates combine complicated query plans with various data sources. Although the LRB workload has a wide complexity range of queries, all deal with one data stream.

In terms of concurrent workload, as the results show the model is a reliable predictor for workloads at MPL range from 2 to 17. Specifically, we can see an exact CRPS and MSE values (i.e. .042 and .010) for memory prediction in both LRB and LRB Mix. However, the MDN has better performance in CPU prediction of LRB compared with LRB Mix. In this regard, the CRPS error reduces as the MDN architecture becomes more complex. This is why the combined workloads are much more complex, hence requires more sophisticated architecture. We repeated the experiment for another workload with 5.5K traces on Amazon EC2, observing similar performance. Due to space limitations, those are not reported.

To compare the proposed approach with the state of the art techniques, we need to treat it as single point estimator and therefore use MSE metric error for comparison. In terms of memory utilization prediction, a closer look at the data indicates that the MDN outperforms the SVM technique in all the experiments. In LRB and TPC-H, the REPTree shows less error, whereas in LRB Mix the opposite observation holds true. When it comes to CPU prediction, our approach is a better resource usage estimator compared with both the REPTree and the SVM in LRB workload. In both LRB Mix and TPC-H, the MDN performance is in between the REPTree and SVM. To be more specific, in LRB Mix, our approach outperforms the REPTree while it shows higher MSE value compared with the SVM.

In summary, our approach outperforms the state of the art single point techniques in 8 out of 12 experiments conducted using the SVM and REPTree. This result is quite promising because it shows that our approach is not only able to predict the full distribution over targets accurately, it is also a reliable single point estimator.

6.3 Training Times and Overhead

In this section we evaluate the training time complexity of the proposed models, as well as the overhead of using them at runtime. Table 5 shows the training times as regards to different workload sizes. As we can see, the training cost is very small and it grows linearly as per the training set size.

Prediction Cost. A crucial issue for the deployment of the resulting estimation models is the overhead of invoking them at runtime. For this purpose, we measured the elapsed time for evaluating an MDN model for a given input feature set on a 2.80GHz Intel Core i7, and obtained an overhead of about 0.2 ms for each call. These numbers show that the MDN is quick enough to become as an integral part of any workload management strategies at runtime.

TABLE 5: Training times in seconds as regards to different workload sizes for $1K$ iterations.

	LRB Mix	LRB Mix_EC2	TPC-H	LRB
Workload Size	$0.5K$	$5.5K$	$8.7K$	$17.2K$
Elapsed Time (s)	3.2	9.65	11.78	22.62

6.4 Follow-up Applications of Distribution-Based Prediction

We presented one of the main applications of our approach in 5.1. To provide a clear picture of what more we can get from the provided prediction technique, we have visualized some sample distribution predictions from a test set of TPC-H workload. Figure 10 plots 14 random sample predicted PDFs for CPU and memory consumption in which they were selected from the model with 3 and 5 GMM components respectively. The histograms of the resource usage of the whole test dataset are also depicted. Each PDF may (not) belong to different queries as they were selected randomly from the test datasets, mean that they are conditioned on different inputs. The dotted vertical line shows the observation value.

As shown in 10(a) and 10(b), the PDFs successfully approximate the resource usage distributions which are within the range [0.1, 0.6] and [0.1, 0.5] for CPU and memory usage respectively. The models for CPU and memory resource usage above the values 0.6 and 0.5 are much more uncertain. In other words, the tendency of all CPU and memory PDFs are to the right hand side of the diagram and this is consistent with the actual resource usage (i.e. plotted histograms) in which we hardly face resource demand above 0.5. Unlike others, the PDFs 2, 3, 13, and 14 are bimodal in which two kernels have comparable priors. This means our model is able to capture the multi-modal nature of the target.

These sample PDFs visually show that the MDN is also a useful classifier in the classic point estimate sense. As regards this point, the CPU PDFs compared to memory ones perform better as the sharpness and the spread of predictive density is more evenly distributed over the target zone. Although the memory PDFs – particularly 8, 11, 12 and 14 – give inaccurate prediction of the target values, they are successful in locating the shape of distributions.

Upper and lower bounds of resource usage simplifies the task of performance isolation since, for example, our predictions in all figures capture the dominant CPU and memory usage precisely. SLA specifications and billing management also become more applicable and reliable for both clients and providers when there is an initial measure of the actual contribution of each workload in terms of the overall resource consumption. When it comes to performance inspection, diagnosing abnormal behavior based on the predicted numbers is also viable. For example, Figure 10 (b) reports that for a given set of queries we will not face peak memory usage (>0.5) very often, hence superior peak

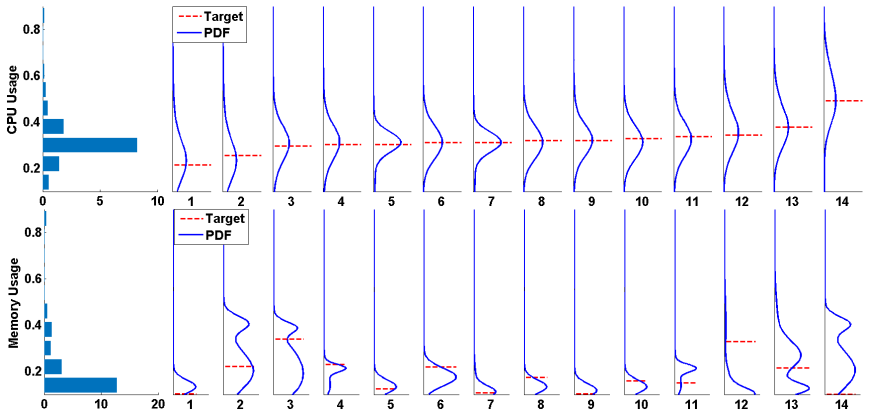


Fig. 10: Sample PDF predictions for (a) CPU and (b) memory usage from TPC-H workload.

memory usage in the live environment signals the presence of a fault in the stream processing system.

7 Conclusions and Future Work

This paper presented a novel approach for resource usage distribution prediction of data stream processing workloads. We demonstrated that the predicted distributions have the potential to become an integral component of the automated workload management systems via developing two novel applications: i) predictable auto-scaling policy setting; and ii) a distribution-based admission controller.

For future work, we plan to enhance the MDN to be able to build prediction models at runtime. For this purpose, with the aid of online learning notions the MDN will be revisited to be able to take an initial guess model and then picks up one-one observation from the training set and recalibrates the weights on each input parameter. Another future plan is considering the security aspects of large scale data processing [20], [28], [29] in the proposed applications.

References

[1] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Modeling and exploiting query interactions in database systems. In *CIKM*, pages 183–192. ACM, 2008.

[2] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik. Learning-based query performance modeling and prediction. In *ICDE*, pages 390–401. IEEE, 2012.

[3] H. Appelrath, D. Geesen, M. Grawunder, T. Michelsen, D. Nicklas, et al. Odysseus: a highly customizable framework for creating efficient event stream management systems. In *DEBS*, pages 367–368. ACM, 2012.

[4] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[5] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: a stream data management benchmark. In *VLDB*, pages 480–491. VLDB Endowment, 2004.

[6] C. Balkesen, N. Tatbul, and M. T. Özsu. Adaptive input admission and management for parallel stream processing. In *DEBS*, pages 15–26. ACM, 2013.

[7] C. M. Bishop. Mixture density networks. 1994.

[8] J. Cervino, E. Kalyvianaki, J. Salvachua, and P. Pietzuch. Adaptive provisioning of stream processing systems in the cloud. In *ICDEW*, pages 295–301. IEEE, 2012.

[9] Y. Chi, H. Hacígümüş, W.-P. Hsiung, and J. F. Naughton. Distribution-based query scheduling. *VLDB*, 6(9):673–684, 2013.

[10] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal. Performance prediction for concurrent database workloads. In *SIGMOD*, pages 337–348. ACM, 2011.

[11] J. Duggan, O. Papaemmanouil, U. Cetintemel, and E. Upfal. Contender: A resource modeling approach for concurrent query performance prediction. In *EDBT*, pages 109–120, 2014.

[12] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE*, pages 592–603. IEEE, 2009.

[13] T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[14] I. J. Good. Rational decisions. *Journal of the Royal Statistical Society.*, pages 107–114, 1952.

[15] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[16] T. Heinze, V. Pappalardo, Z. Jerzak, and C. Fetzer. Auto-scaling techniques for elastic data stream processing. In *ICDEW*, pages 296–302. IEEE, 2014.

[17] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm. A catalog of stream processing optimizations. *ACM Computing Surveys (CSUR)*, 46(4):46, 2014.

[18] J. Li, A. C. König, V. Narasayya, and S. Chaudhuri. Robust estimation of resource consumption for sql queries using statistical techniques. *VLDB*, 5(11):1555–1566, 2012.

[19] H. C. Lim, S. Babu, and J. S. Chase. Automated control for elastic storage. In *ICAC*, pages 1–10. ACM, 2010.

[20] C. Liu, et al. Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates. TPDS 25.9 (2014): 2234-2244.

[21] B. Mozafari, C. Curino, A. Jindal, and S. Madden. Performance and resource modeling in highly-concurrent oltp workloads. In *SIGMOD*, pages 301–312. ACM, 2013.

[22] I. Nabney. *NETLAB: algorithms for pattern recognition*. Springer Science & Business Media, 2002.

[23] W. Wu, X. Wu, H. Hacígümüş, and J. F. Naughton. Uncertainty aware query execution time prediction. *VLDB*, 7(14):1857–1868, 2014.

[24] W. Wu, Y. Chi, H. Hacígümüş, and J. F. Naughton. Towards predicting query execution time for concurrent and dynamic database workloads. *VLDB*, 6(10):925–936, 2013.

[25] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigumus, and J. F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *ICDE*, pages 1081–1092. IEEE, 2013.

[26] P. Xiong, Y. Chi, S. Zhu, H. Moon, C. Pu, and H. Hacigumus. Smartsla: Cost-sensitive management of virtualized resources for cpu-bound database services. *TPDS*, 2014.

[27] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. HacigümüŞ. Activesla: a profit-oriented admission control framework for database-as-a-service providers. In *SoCC*, page 15. ACM, 2011.

[28] X. Zhang, et al. A hybrid approach for scalable sub-tree anonymization over big data using MapReduce on cloud. JCSS 80.5 (2014): 1008-1020.

[29] C. Yang, et al. A spatiotemporal compression based approach for efficient big data processing on cloud. JCSS 80.8 (2014): 1563-1583.