PerfEnforce Demonstration: Data Analytics with Performance Guarantees

Jennifer Ortiz, Brendan Lee, Magdalena Balazinska
Computer Science and Engineering Department, University of Washington
Seattle, Washington, USA
{jortiz16, lee33, magda}@cs.washington.edu

ABSTRACT

We demonstrate PerfEnforce, a dynamic scaling engine for analytics services. PerfEnforce automatically scales a cluster of virtual machines in order to minimize costs while probabilistically meeting the query runtime guarantees offered by a performance-oriented service level agreement (SLA). The demonstration will show three families of dynamic scaling algorithms –feedback control, reinforcement learning, and online machine learning– and will enable attendees to change tuning parameters, performance thresholds, and workloads to compare and contrast the algorithms in different settings.

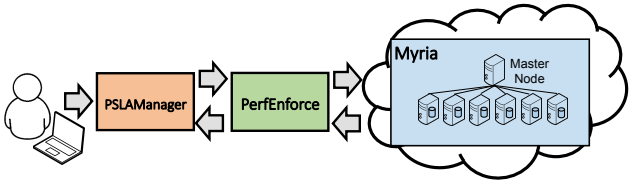

Figure 1: PerfEnforce deployment: PerfEnforce sits on top of an elastically scalable big data management system (*e.g.*, Myria in the demonstration) in support of performance-oriented SLAs for cloud data analytics (*e.g.*, PSLAManager in the demonstration).

1. INTRODUCTION

A variety of data analytics systems are available as cloud services today, including Amazon Elastic MapReduce (EMR) [1], Redshift [1], Azure's HDInsight [2], and others. With these services, users have access to compute clusters that come pre-packaged with data analytics tools. With most of these services, users select and pay for a given cluster configuration: *i.e.*, number and type of service instances. It is well known, however, that users often have difficultly selecting configurations that meet their needs. Frequently, users need to test many configurations before finding a suitable one [6]. Some database cloud services such as Amazon RDS [1] and Azure [2] offer the ability to scale a database application automatically, but they require users to manually specify the scaling conditions, which requires deep expertise and planning. Recent work offers solutions for automatic cluster resizing but either focuses on transaction processing [8, 3] or requires a known and profiled workload [6, 10].

An alternate approach is to enable users to purchase not a cluster size but a performance level. In Personalized Service Level Agreements [12], we developed an approach where users purchase service tiers with query time guarantees. Similarly, the XCloud system [13] supports SLAs with customer-specific performance criteria. The challenge behind selling performance-focused SLAs for data analytics is in guaranteeing the query runtimes advertised in the SLAs.

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA

ISBN 978-1-4503-3531-7/16/06...$15.00

DOI: http://dx.doi.org/10.1145/2882903.2899402

We design the PerfEnforce system to address the above need. PerfEnforce works in support of performance-centric SLAs for data analytics services. PerfEnforce targets services such as EMR or Redshift, where each user runs the service in his or her own cluster of virtual machines. Figure 1 shows the overall system architecture. In our approach, the user purchases a service tier with an SLA that specifies query runtimes. These runtimes correspond to query time estimates for specific cluster sizes, which define the tiers of service. Once the user selects a service tier, the cloud service instantiates the corresponding cluster. As the user executes queries, prediction inaccuracies and interference from other tenants using the service can cause query times to differ from the estimated ones purchased by the user. To meet the terms of the performance-based SLA, PerfEnforce automatically re-sizes the cluster allocated to the user. PerfEnforce seeks to minimize the cluster size allocated to the user subject to probabilistically satisfying the query time guarantees in the SLA. In the demonstration, we use the PSLAManager [12] to generate performance-based SLAs and the Myria system [5] as the elastically scalable data processing engine.

Several systems have recently studied performance guarantees through dynamic resource allocation in storage systems [9] using feedback control, or in transaction processing systems [8] using reinforcement learning. In this demonstration, we show how applying these techniques enables PerfEnforce to effectively scale the cluster by adding or removing nodes. In PerfEnforce, we also develop a third technique based on online machine learning. In contrast to feedback control and reinforcement learning, which are *reactive* methods, online machine learning enables PerfEnforce to change the cluster size *before* running an incoming query, which can improve QoS. As the user runs more queries in the session, this approach continuously improves the query runtime prediction model. In addition, it can compensate for early prediction errors using a control technique. This demonstration will show how the three techniques trade off QoS and cost given different databases and query workloads as exemplified in Figure 3.

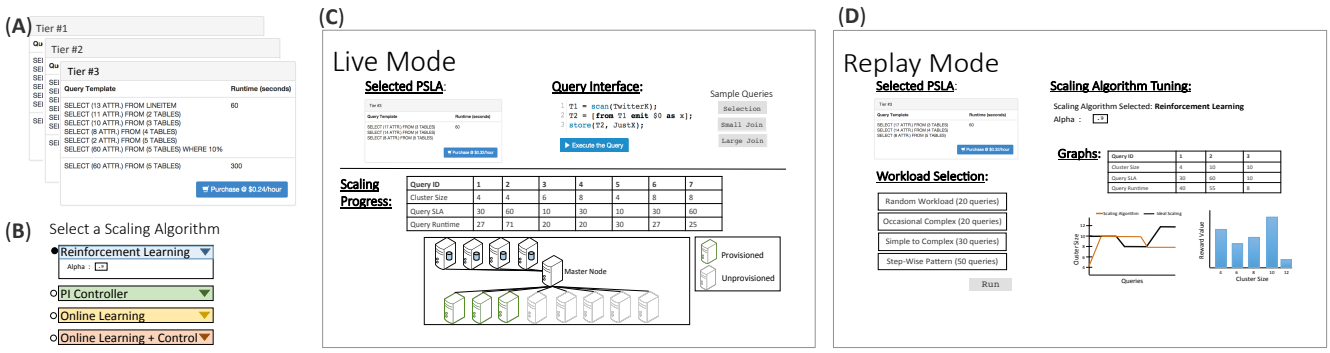

Figure 2: Demonstration screenshot described in Section 2.

The demonstration will enable attendees to experiment with the three cluster-scaling algorithms and experience their benefits and limitations. The attendees will select a performance agreement, a query workload, and the scaling algorithm. They will then observe how the selected algorithm changes the cluster size dynamically and the resulting query performance. Attendees will also be able to adjust tunable parameters for these algorithms. The demonstration will run as an Amazon EC2 cluster in one of two modes: *live* and *replay*. In the live mode, PerfEnforce will add and remove virtual machines (VMs) as the system executes queries on real data. The replay mode will enable the demonstration of PerfEnforce on longer queries and bigger workloads by showing only the cluster resizing decisions without actually re-sizing the cluster nor executing queries. Replay will use previously recorded query runtimes.

The demonstration will make two contributions: It will show PerfEnforce in action running queries on real data in the Amazon EC2 cloud. It will also enable attendees to experience different auto-scaling algorithms including feedback control and reinforcement learning used in prior work, as well as PerfEnforce's new approach based on a combination of online machine learning and feedback control.

2. DEMONSTRATION DETAILS

The demonstration starts with 10GB of TPC-H Star Schema Benchmark (TPC-H SSB) data [11] stored in Amazon S3 [1].

- *Selecting a Performance-Centric SLA:* The attendee first selects a service tier available in the SLA generated by our PSLAManager system for the TPC-H SSB dataset as shown in Figure 2 (A). Each service tier corresponds to a configuration of the service and comes with a price, representing the cost of the underlying cluster configuration. The query templates in each tier are grouped by time thresholds, representing the runtime guarantee for those queries. The demonstration uses a set of 5 tiers representing configurations of 4, 6, 8, 10 and 12 workers.

- *Scaling Algorithm:* The user selects and tunes the scaling algorithm that PerfEnforce will use as shown in Figure 2 (B). Available scaling algorithms include reinforcement learning, feedback control, online machine learning and online machine learning combined with feedback control. The scaling algorithm affects the number of queries that meet their SLA and the cloud cost of executing queries.

After making these selections, the user can then execute queries in either *live* or *replay mode*.

Live Mode: In this mode, the user experiences PerfEnforce as a live service. Starting from the purchased configuration, PerfEnforce executes queries and resizes the cluster by spinning up and shutting down service instances. As shown in Figure 2 (C), the user sees three screens: the chosen SLA tier, a query interface to write new queries or select pre-defined queries, and a visualization of the current cluster size including a table describing the query runtime, the SLA runtime, and the cluster size for the current and past queries.

Replay Mode: In the Replay Mode, the user sees four screens. The first screen shows the selected SLA tier. The second one, enables the user to select a pre-defined query workload. For example, one workload runs small queries followed by more complex ones. Another workload is a sequence of random queries. Once the user picks a workload, the system "runs" the queries by reusing their previously recorded runtimes. For each replayed query, the bottom right screen shows a table of the (previously measured) real query runtime, the SLA runtime, and the cluster size. The last screen also shows a set of graphs that vary with the selected scaling technique. If the user selects reinforcement learning, the graphs dynamically show how the reward changes over time for each cluster state. For feedback control, the graphs show the error for each current window as well as the history. For online machine learning, the interface shows predictions of the model compared to the real runtime of the query. In the tuning screen, the user can tune the scaling parameters again if desired.

3. THE PERFENFORCE SYSTEM

Figure 1 shows how PerfEnforce interacts with the other components of a cloud data analytics service. When a user begins her query session, she first purchases a performance tier from the cloud service provider. In our implementation, we use the PSLAManager system [12] to generate these performance tiers.

The PSLAManager from our prior work helps users select a service configuration by displaying a personalized service level agreement (PSLA). A PSLA is based on the user's data (the user provides the schema and basic statistics). Each PSLA tier corresponds to a cluster configuration but the user only sees an hourly price, a set of query templates, and a set of query runtimes. Intuitively, if a tier cost is cheaper, the configuration is smaller, and query latencies are larger. PerfEnforce can be extended to work with other SLA managers.

Once the user purchases a tier of service, PerfEnforce spins up the corresponding cluster and ingests the user's data. For this work, we use Myria as our data analytics cloud service. Myria is a shared-nothing parallel database system [5]. The Myria system has a mas-

ter node that is responsible for sending queries to the worker nodes. The worker nodes ingest the data such that the first four workers each contain one quarter of the data. The first six workers each have one sixth of the data and so on. Each worker stores its data locally using an independent relational management system (RDBMS) instance for local storage. To scale the cluster dynamically, PerfEnforce schedules queries on the first N of the 12 workers. The remaining workers are not used and can be stopped but they need to preserve the locally stored data in case the cluster needs to grow.

As the user executes queries, the responsibility of the PSLAManager is to notify PerfEnforce about the query time guarantee associated with each user query to drive PerfEnforce's cluster re-sizing decisions.

3.1 Problem Description

PerfEnforce's goal is to minimize the number of cluster instances subject to probabilistically meeting the query runtime guarantees of the SLA tier selected by the user. The challenge is that query runtimes in SLAs are based on estimates, which are not always accurate due to cardinality estimation errors, differences between the training and user datasets, and limitations of the features used in the model. Query runtime prediction is not a contribution of our work. Instead, we focus on scaling the system in order to meet the guarantees in the selected SLA.

More formally, given a query session Q, with queries q_0 through q_n and a cluster C, PerfEnforce trades-off two competing goals: quality of service (QoS) and cost. We define cost as $\sum_{q=0}^{n}(\text{size}(\text{C}_\text{q}))$, where $\texttt{size}(C_q)$ is the number of instances in the cluster used to execute query q. We define QoS as $\frac{1}{n}\sum_{q=0}^{n} g(q_i)$ where $g(q) = \{1 \ if \ t_{sla}(q) \geq t_{real}(q) \ ; \ 0 \ otherwise\}$. Here, $t_{sla}(q)$ and $t_{real}(q)$ are the SLA and actual runtimes of a query q, respectively.

QoS and cost are at odds with each other and their combination defines the SLA offered to the user. In this demonstration, instead of picking a specific SLA level (*e.g.*, 90% of queries should meet their SLA), we demonstrate how well different cluster-scaling algorithms can operate the cluster as close as possible to a point where resources are neither wasted nor missing: For each query, we compute the ratio between $t_{real}(q)$ and $t_{sla}(q)$, $\frac{t_{real}(q)}{t_{sla}(q)}$, and strive to achieve a ratio as close to 1 as possible. If this ratio is above 1, the cluster is underprovisioned. A ratio below 1 indicates that the cluster is in an overprovisioned state. In the next section, we describe how each algorithm effectively attempts to reach this goal.

3.2 Scaling Algorithms

We demonstrate the following four cluster-scaling algorithms.

3.2.1 *Reactive Scaling Algorithms*

We first describe reactive scaling algorithms. These algorithms take action after they witness either a good or bad event. In PerfEnforce, we implement proportional integral control and reinforcement learning as our reactive methods.

Proportional Integral Control Feedback control [7] is a commonly used approach to regulate a system in order to ensure that it operates at a given reference point. This approach has successfully been applied in various contexts including dynamically scaling data storage engines [9]. We use a proportional-integral controller (PI) as a method that helps PerfEnforce react based on the magnitude of the error while avoiding oscillations over time.

At each time step, t, the controller produces an actuator value $u(t)$ that causes the system to produce an output $y(t+1)$ at the next time step. The goal is for the system output $y(t)$ to be equal to some desired reference output $r(t)$. In an integral controller, the actuator value depends on the accumulation of past errors of the system. This can be represented as:

$$u(t+1) = u(t) + k_i e(t) \tag{1}$$

where $e(t) = y(t) - r(t)$, with $r(t)$ being the target system output and $y(t)$ being the observed output. k_i represents the gain of the integral control. Ideally, this parameter is tuned in such a way that helps drive $e(t)$ to 0.

In our scenario, the actuator value $u(t)$ is the discrete number of VMs provisioned. It is initialized to the cluster size corresponding to the SLA tier that the user purchased.

As for the system output, $y(t)$, we use the average ratio of the real query runtime $t_{real}(q)$ over the query runtime promised in the SLA, $t_{sla}(q)$, over some time window of queries $w(t)$.

$$y(t) = \frac{1}{|w(t)|}\sum_{q \in w(t)} \frac{t_{real}(q)}{t_{sla}(q)} \tag{2}$$

where $|w(t)|$ is the number of queries in $w(t)$.

Our target operating point is thus $r(t) = 1.0$ and the error $e(t) = y(t) - r(t)$ captures a percent error between the current and desired average runtime ratios. Because the number of VMs to add and remove given such a percent error depends on the cluster size, we add that size to the error computation as follows: $e(t) = (y(t) - r(t))u(t)$.

Integral control alone may be slow to react to changes in the workload. Therefore, we also introduce a proportional control component, to yield a PI controller with the following formulation:

$$u(t+1) = u(0) + \sum_{x=0}^{t} k_i e(x) + k_p e(t) \tag{3}$$

Reinforcement Learning As our second reactive method, we use reinforcement learning (RL). This approach has successfully been applied in the TIRAMOLA system, which supports elastic scaling of NoSQL databases [8].

With reinforcement learning, we model cluster scaling as a Markov decision process, where at each state s_1, the model makes a probabilistic decision to move to another state, s_2. The goal is to move to a state with the highest reward, $R(s)$. As the system moves between states, the model learns and updates the values of the rewards. In our setting, each cluster configuration represents a state in the model. We define the reward function to be the real-to-SLA runtime ratio. Our goal is to favor states with the reward closest to 1.0, where the real query runtimes are closest to the SLA runtimes (see Section 3.1). In RL, every time the system transitions to a state s, it updates the reward function for that state. In our setting, we use the following equation, where $R(s')$ denotes the updated reward for state s:

$$R(s') = \alpha * (\frac{t_{real}(q)}{t_{sla}(q)} - R(s)) + R(s) \tag{4}$$

α is the learning rate, which controls how fast learning takes place: *i.e.*, how much to update the reward based on the observed runtime value, $t_{real}(q)$, for the latest query q executed in state s.

At the initialization of the model, every state must begin with a defined reward value, $R(s)$, in order to determine which state to move to. This implies that the system must have prior knowledge about the performance of real query runtimes for each configuration. In PerfEnforce, we choose to make no assumptions about runtimes and initialize the reward for each state to 1.0. However, we do know that reward values will be lower for larger cluster sizes. To capture this information, we maintain a set of states called *active*

states. When the query session begins, *active states* only contains the cluster size purchased by the user. If the reward for the current state goes above 1.0, we add the next larger cluster size to the active states, unless it was already in that set. If the reward for the current state goes below 1.0, we similarly add the next smaller cluster size. We repeat the process until all cluster sizes have been added.

3.2.2 *Proactive Scaling Algorithms*

Instead of approaches that react to runtime errors such as the PI controller and reinforcement learning, we also explore an approach that makes use of a predictive model. For each incoming query, PerfEnforce predicts the runtime for the query for each configuration and switches to the cheapest configuration that meets the SLA.

PerfEnforce first builds an offline model. For training data, we use the Parallel Data Generation Framework tool [14] to generate a 10GB dataset with a set of 2500 queries. Training data consists of query plan features including the estimated max cost, estimated number of rows, estimated width, and number of workers.

We observe that we can significantly improve this offline model if we incorporate information about specific queries that the user executes on his data. We achieve this goal by using an online machine learning model: as the user executes queries, PerfEnforce improves the model in an online fashion. We use the MOA (Massive Online Analysis) tool for online learning [4].

Online Learning (OL). We use the perceptron algorithm for the online model. The perceptron algorithm works by adjusting model weights for each new data point. We find that it adapts more quickly to new information than an active-learning based approach. PerfEnforce initiates the perceptron model by learning from the training set. For the first query, PerfEnforce uses this model to predict a runtime for each possible configuration. The cluster size with the closest runtime to the query's SLA is chosen. Once the system runs the query and knows the real runtime, it adds this information to the model.

Online Learning with Control Buffer We find that it takes a few queries before perceptron is able to start improving prediction accuracy. We find that we can not simply increase the learning rate as it causes significant oscillations in the predictions. Instead, to ensure high quality of service even early on in a query session, we add a buffer based on the observed prediction errors. We call this method online learning with control buffer (OL+B). We predict the runtime of the next query $t_{pred}(q_{t+1})$ as follows:

$$\begin{aligned} t_{pred}(q_{t+1}) =& \text{Perceptron}(q_{t+1}) \\ &+ k_p * Avg_{wt}(\max(0, \text{PercentError}(t_{real}(q_t), \\ &\text{Perceptron}(q_t))) \end{aligned} \tag{5}$$

Where $\text{Perceptron}(t+1)$ is the runtime prediction for query $t_{pred}(q_{t+1})$. PerfEnforce adjusts that runtime if the previous window of queries resulted in a positive percent error. Based on this adjusted prediction, PerfEnforce allocates the cluster size with the predicted runtime closest to the SLA deadline.

3.2.3 *Comparing Scaling Algorithms*

Figure 3 shows the average performance of each algorithm on three sets of random query workloads. Each workload contains 100 queries. On the x-axis, we measure a normalized value of money saved. This value is derived from the cost for each query, C_q. We calculate the money saved compared with using the largest, 12-node cluster, for each query. On the y-axis, we measure the percent of queries that meet their deadline, essentially showing the *QoS*.

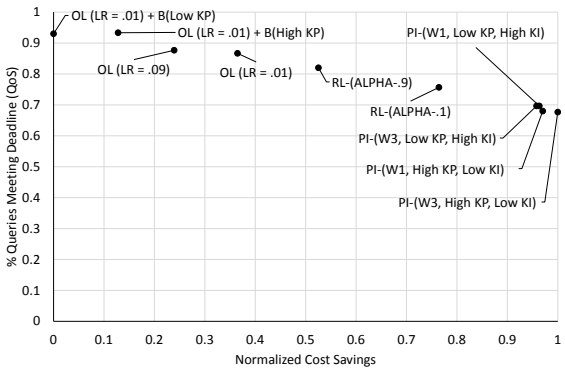

Figure 3: Example Trade-off between Scaling Algorithms

As the graph shows, no method clearly dominates the others. For Proportional Integral Control (PI), we vary the query window sizes, W, and use a combination of either high or low values for the gain parameters. From these experiments, higher k_i values result in a slightly higher QoS than using higher k_p. For reinforcement learning (RL), we end up with a higher QoS (75% - 82%) than the PI control methods. We find that Online learning (OL) yields the highest QoS (above 85%) for several learning rates, LR. Finally, Online Learning with Control Buffer(OL+B), has better QoS compared to OL but at an even higher cost. In this experiment, both OL and OL+B techniques provide the highest QoS, while PI control leads to largest cost savings.

4. CONCLUSION

In this demonstration, we introduce the PerfEnforce system. Based on the user's performance-centric SLA, PerfEnforce scales the system in order to achieve good QoS at a low cost. PerfEnforce can scale the system using a variety of scaling algorithms. In this demonstration, the user will observe and tune the algorithms on a running cloud service.

Acknowledgements This project was supported in part by NSF grants IIS-1247469 and IIS-1524535, gifts from Amazon, the Intel Science and Technology Center for Big Data, and Facebook. J. Ortiz was supported in part by an NSF graduate fellowship. We also thank Joseph L. Hellerstein for insightful discussions.

5. REFERENCES

[1] Amazon AWS. http://aws.amazon.com/.
[2] Microsoft azure. http://azure.microsoft.com/en-us/.
[3] S. Barker et al. Shuttledb: Database-aware elasticity in the cloud. In *ICAC '14*.
[4] A. Bifet et al. Moa: Massive online analysis. In *Journal of Machine Learning Research*, 2010.
[5] D. Halperin et al. Demonstration of the Myria big data management service. In *SIGMOD*, 2014.
[6] H. Herodotou et al. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proc. of the 23rd SOSP Symp.*, 2011.
[7] P. K. Janert. *Feedback Control for Computer Systems*. O'Reilly Media, Inc., 2013.
[8] I. Konstantinou et al. TIRAMOLA: elastic nosql provisioning through a cloud management platform. In *Proc. of the SIGMOD Conf.*, 2012.
[9] H. Lim et al. Automated control for elastic storage. In *ICAC 2010*.
[10] H. A. Mahmoud et al. Cloudoptimizer: multi-tenancy for i/o-bound OLAP workloads. In *edbt13*, 2013.
[11] P. O'Neil, E. O'Neil, and X. Chen. The star schema benchmark. http://www.cs.umb.edu/~poneil/StarSchemaB.PDF.
[12] J. Ortiz et al. Changing the face of database cloud services with personalized service level agreements. 2015.
[13] O. Papaemmanouil. Supporting extensible performance slas for cloud databases. In *Proc. of the 28th ICDE Conf.*, 2012.
[14] T. Rabl et al. A data generator for cloud-scale benchmarking. TPCTC'10. Springer-Verlag.