PerfEnforce Overview: A Dynamic Scaling Engine for Analytics with Performance Guarantees

Jennifer Ortiz

Computer Science and Engineering Department, University of Washington
Seattle, Washington, USA
jortiz16@cs.washington.edu

1. PROBLEM AND MOTIVATION

A variety of systems for data analytics are available as cloud services today, including Amazon Elastic MapReduce (EMR), Amazon Redshift, Azure's HDInsight, and several others. While these services greatly facilitate access to compute resources and data analytics software, they remain hard for users to tune in terms of cost and performance. Users choose a price-performance trade-off by selecting a desired number and type of service instances. It is well-known, however, that users have difficulty determining their resource needs and often attempt many configurations before finding a suitable one.

Performance-centric service level agreements (SLAs) have recently been proposed in response to the above limitations. In our previous work, we've developed *Personalized Service Level Agreements* (PSLAs), an approach where users purchase service tiers (representing different performance levels) with query time guarantees. However, the challenge behind selling performance-focused SLAs for data analytics is in guaranteeing the query runtimes advertised in the SLAs.

To address this problem, we developed the PerfEnforce system. PerfEnforce works with a cloud service to meet the goals of a performance-based SLA. PerfEnforce is designed for shared-nothing data management systems (*e.g.*, Myria, Spark, Impala, EMR) supporting data analytic workloads. In Figure 1, we show how PerfEnforce interacts with the PSLAManager system to provide SLAs with performance guarantees. As such, PerfEnforce is designed to support SLAs that guarantee specific query runtimes. 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, the system automatically resizes the cluster allocated to the user. PerfEnforce seeks to minimize the cluster size subject to satisfying the query runtime guarantees in the SLA. A second challenge is determining when resources should be terminated or whether new ones should be provisioned. In a cloud setting, provisioning resources is not trivial. Turning resources on or off comes with a time delay that could potentially impact tenant SLAs. In addition, user behav-

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

ior is relatively unknown to the system. For example, a user might end a query session at any time, or delay before submitting the next query. At the start of the query session, PerfEnforce is not aware about the types of queries that will be submitted into the system. We focus on techniques that could probabilistically help determine the number of machines that should be provisioned.

2. BACKGROUND AND RELATED WORK

Performance Guarantees in Data Analytics Performance guarantees have traditionally been the focus of real-time database systems [13], where the goal is to schedule queries in a fixed-size cluster to ensure they meet their deadlines. More recently, dynamic provisioning and admission control methods have enabled OLAP and OLTP systems to make profitable choices with respect to performance guarantees [3, 4, 23], possibly postponing or even simply rejecting queries. PerfEnforce's goal is to scale the cluster with minimal delay to meet SLA guarantees as closely as possible.

Multi-Tenant Performance Guarantees An active area of research in multi-tenant cloud DBMS systems is *tenant packing* [7, 16, 15], or how best to colocate tenants on a shared set of machines or even DBMS instances. In contrast, we focus on the independent database user who spins up a private cluster in the cloud. We seek to minimize the size of that cluster while meeting SLA runtime guarantees.

Query Runtime Prediction Previous work has relied on classification and regression techniques to determine whether a query will miss or meet a deadline [23], building gray-box performance models [9], using historical traces of previous workloads [8] or running smaller samples of the workload with a low overhead [21]. Most closely related is work by Herodotou et. al. [10], which utilizes a previously profiled workload from the user in order to predict the runtime of that program against different sized clusters. Work by Jalaparti et. al. [11] focuses on generating resource combinations given performance goals from the user. Instead of building a white-box or analytical model, we focus on using a model that does not require an extensive understanding of a single system. We also look for models that can quickly adapt based on changing system conditions. In addition, we focus on interactive, ad-hoc queries for which there are no existing profiles.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGMOD '17 Student Research Competition, May 14–19, 2017, Chicago, IL, USA.

© 2017 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4199-8/17/05.

DOI: <http://dx.doi.org/10.1145/3055167.3055175>

Figure 2: Distributions of $\frac{t_{real}(q_{ij})}{t_{sla}(q_{ij})}$ ratios for each technique on a random query workload.

Elasticity Cloud providers offer the ability to scale a database application [1, 2]. However, they require users to manually specify scaling conditions through vendor-specific APIs. This requires expertise and imposes the risk of resource over-provisioning. Moreover, these scaling features can be costly, as some of these actions are subject to service downtimes and may take several minutes to complete (such as data rebalancing) [1]. Most academic work on elastic systems focuses on OLTP workloads [5, 20, 22] and thus develops new techniques for tenant database migration [6], data re-partitioning while maintaining consistency [18], or automated replication [22]. In these systems, the goal is to maximize aggregate system performance, while our focus is on per-query performance guarantees.

3. APPROACH AND UNIQUENESS

The goal of PerfEnforce is to minimize resource utilization subject to meeting guaranteed query runtimes. Several systems have recently studied performance guarantees through dynamic resource allocation in storage systems by using *feedback control*, or in OLAP systems using *reinforcement learning*. With PerfEnforce, we show how applying these techniques enables the system to effectively scale the cluster by adding or removing nodes. We also study and apply a third technique based on *deep learning*.

3.1 PerfEnforce Goal

Each user, u_i , submits a sequence of queries within their respective query session. We define the sequence of queries issued for user u_i as SEQ_i . This set holds j ordered pairs, where each pair contains a query q_{ij} and its corresponding SLA $t_{sla}(q_{ij})$. We define this set as $SEQ_i = \{(q_{i1}, t_{sla}(q_{i1})), \dots, (q_{ij}, t_{sla}(q_{ij}))\}$.

In PerfEnforce, SEQ_i is not known a priori since the system does not assume that each user has a pre-defined query workload. We model the arrival of queries as a Poisson process and define the parameter λ_{think} to represent a distribution of average user *think times*. *Think time* is the amount of time a user spends observing a query result before submitting the next query. In addition, a user may decide to finish a query session at any time. We include this uncertainty also as a Poisson process where the parameter $\lambda_{terminate}$ describes a distribution of average *query session lengths*. Given these parameters, we characterize a query session for user i as: $Q_i = (SEQ_i, \lambda_{think}, \lambda_{terminate})$.

To meet each query’s SLA, PerfEnforce must answer two key questions: 1) When to scale the system? and 2) What configuration to scale to? Given each query j in Q_i , we define a *query performance ratio* as $\frac{t_{real}(q_{ij})}{t_{sla}(q_{ij})}$ where $t_{sla}(q_{ij})$ and $t_{real}(q_{ij})$ represent the SLA and actual runtimes of a query q_{ij} , respectively. Throughout a query session, the system will scale according to a given scaling technique S , and will accumulate a distribution of *query performance ratios* which we define as $D_{Q_i, S}$. To neither waste cluster resources or violate SLA runtimes, it is ideal to target performance ratios as close to 1.0 as possible. The best cluster scaling algorithm is one that yields a tight distribution (a small σ) and a mean (μ) close to 1.0 for all queries in the session, which ensures that most query runtimes stay close to the SLA runtimes.

3.2 PerfEnforce Scaling Techniques

We look at scaling algorithms that can quickly adapt and select the best cluster configuration that will meet each query’s SLA. We compare against existing scaling algorithm techniques including proportional integral control and reinforcement learning.

Proportional Integral Control Feedback control [12] is commonly used to regulate a system in order to ensure that it operates at a given reference point. We use a proportional-integral controller (PI) as a method that helps PerfEnforce react based on the magnitude of the SLA error while avoiding oscillations over time.

Reinforcement Learning We also study how to use reinforcement learning as a scaling algorithm. This approach has successfully been applied in the TIRAMOLA system, which supports elastic scaling of NoSQL databases [14]. We implement a contextual multi-armed bandit model (CMAB) as seen in work by Marcus *et al.* [17]. In a CMAB model, there is context X and a set of actions A . When choosing an action $a \in A$, we observe a reward r . The goal of this approach is to choose the action that will lead to the highest reward. Within our setting, X represents the set of features about the incoming query, A represents the set of possible cluster sizes PerfEnforce can scale to and r represents the ratio between the query’s real runtime and SLA. The cluster size with a reward closest to 1.0 is chosen, as it represents the cluster size the model believes is closest to the SLA.

Deep Learning PerfEnforce first builds an offline neural network model for a given cloud data analytics service. We use the Parallel Data Generation Framework tool [19] to help generate a training dataset. When a user first issues a query, the neural network learns from the training set and predicts the runtime for the user’s query on each configuration size. The query is scheduled on the cluster size predicted to execute at a runtime close to the SLA. Initially, the model is expected to be inaccurate, as it does not know much about the user’s dataset or query workload. However, as the user executes queries, the model improves in an online fashion by feeding information about the query back into the neural network.

4. RESULTS AND CONTRIBUTIONS

In Figure 2, we show the resulting distribution of ratios on all scaling techniques for a random query workload. To help compare, we include an *Oracle* which is an additional “all-knowing” technique that executes each query at the best cluster configuration size (the one closest to the SLA). We also include a technique that randomly selects between different cluster sizes, shown as *Random*. From these results, we can observe that the deep learning technique provides a distribution that is very similar to the Oracle. We have also found that using this technique results in a model that 1) improves performance predictions over time 2) is resilient to changing query workloads and 3) can adapt runtime predictions depending on changing system conditions (i.e. warm cache, system performance degradation, etc.).

Acknowledgements This project was supported in part by NSF grants IIS-1247469 and IIS-1524535, gifts from Amazon, the ISTC for Big Data, Facebook and the NSF GRFP.

5. REFERENCES

- [1] Amazon AWS. <http://aws.amazon.com/>.
- [2] Microsoft Azure. <http://azure.microsoft.com/en-us/>.
- [3] Y. Chi et al. iCBS: Incremental cost-based scheduling under piecewise linear SLAs. *PVLDB*, 4(9):563–574, 2011.
- [4] Y. Chi et al. SLA-tree: a framework for efficiently supporting SLA-based decisions in cloud computing. In *Proc. of the EDBT Conf.*, pages 129–140, 2011.
- [5] S. Das et al. Albatross: Lightweight elasticity in shared storage databases for the cloud using live data migration. *PVLDB*, 4(8):494–505, 2011.
- [6] A. J. Elmore et al. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In *Proc. of the SIGMOD Conf.*, pages 301–312, 2011.
- [7] A. J. Elmore et al. Characterizing tenant behavior for placement and crisis mitigation in multitenant DBMSs. In *Proc. of the SIGMOD Conf.*, pages 517–528, 2013.
- [8] A. D. Ferguson et al. Jockey: guaranteed job latency in data parallel clusters. In *EuroSys '12*.
- [9] A. Gandhi et al. Autoscaling for hadoop clusters. In *IC2E 2016*.
- [10] H. Herodotou et al. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proc. of the Second SoCC Conf.*, page 18, 2011.
- [11] V. Jalaparti et al. Bridging the tenant-provider gap in cloud services. In *Proc. of the 3rd ACM Symp. on Cloud Computing*, page 10, 2012.
- [12] P. K. Janert. *Feedback Control for Computer Systems*. O'Reilly Media, Inc., 2013.
- [13] B. Kao et al. An overview of real-time database systems. Prentice-Hall, Inc., 1995.
- [14] I. Konstantinou et al. TIRAMOLA: elastic NoSQL provisioning through a cloud management platform. In *Proc. of the SIGMOD Conf.*, pages 725–728, 2012.
- [15] Z. Liu et al. PMAx: Tenant placement in multitenant databases for profit maximization. In *Proc. of the EDBT Conf.*, pages 442–453, 2013.
- [16] H. A. Mahmoud et al. CloudOptimizer: multi-tenancy for I/O-bound OLAP workloads. In *Proc. of the EDBT Conf.*, pages 77–88, 2013.
- [17] R. Marcus et al. Releasing cloud databases from the chains of performance prediction models. In *CIDR*, 2017.
- [18] U. F. Minhas et al. Elastic scale-out for partition-based database systems. *ICDEW 2012*, pages 281–288.
- [19] T. Rabl et al. A data generator for cloud-scale benchmarking. *TPCTC'10*, pages 41–56.
- [20] M. Stonebraker et al. The VoltDB main memory DBMS. *IEEE Data Eng. Bull.*, 36, 2013.
- [21] S. Venkataraman et al. Ernest: Efficient performance prediction for large scale advanced analytics. In *NSDI 2016*.
- [22] H. T. Vo et al. Towards elastic transactional cloud storage with range query support. *PVLDB*, 3(1):506–517, 2010.
- [23] P. Xiong et al. ActiveSLA: a profit-oriented admission control framework for database-as-a-service providers. In *Proc. of the Second SoCC Conf.*, page 15, 2011.