A Vision for Personalized Service Level Agreements in the Cloud

Jennifer Ortiz[†], Victor Teixeira de Almeida[†,§], Magdalena Balazinska[†]
[†]Computer Science and Engineering Department, University of Washington
Seattle, Washington, USA
[§]PETROBRAS S.A., Rio de Janeiro, RJ, Brazil
{jortiz16, valmeida, magda}@cs.washington.edu

ABSTRACT

Public Clouds today provide a variety of services for data analysis such as Amazon Elastic MapReduce and Google BigQuery. Each service comes with a pricing model and service level agreement (SLA). Today's pricing models and SLAs are described at the level of compute resources (instance-hours or gigabytes processed). They are also different from one service to the next. Both conditions make it difficult for users to select a service, pick a configuration, and predict the actual analysis cost. To address this challenge, we propose a new abstraction, called a *Personalized Service Level Agreement*, where users are presented with what they can do with their data in terms of query capabilities, guaranteed query performance and fixed hourly prices.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

Management, Performance

1. INTRODUCTION

Many data management systems today are available as Cloud services. For example, Amazon Web Services (AWS) [1] include the Relational Database Service (RDS) and Elastic MapReduce (EMR); Google offers BigQuery [5]; and SQL Server is available on Windows Azure [2]. Each service comes with a pricing model that indicates the price to pay based on the level of service. In this paper, we address the challenges behind selecting a service and a desired configuration for that service.

An important challenge with today's pricing models is that they force users to translate their data management needs into resource needs (How many instances should I use? How many gigabytes will my queries process?). There is thus a disconnect between the resource-centric approach expressed by Cloud providers and what the users actually wish to acquire [10]. The knowledge required to understand the resources needed for data management workloads is a challenge – particularly when a user does not always have a clear understanding of their data or even know what they are looking for [11]. Furthermore, pricing models can be wildly different across providers [13]. For example, AWS charges for the number and size of compute instances, while BigQuery charges per GB processed by a query. This heterogeneity complicates the decision behind selecting a service.

DanaC'13, June 23, 2013, New York, NY, USA

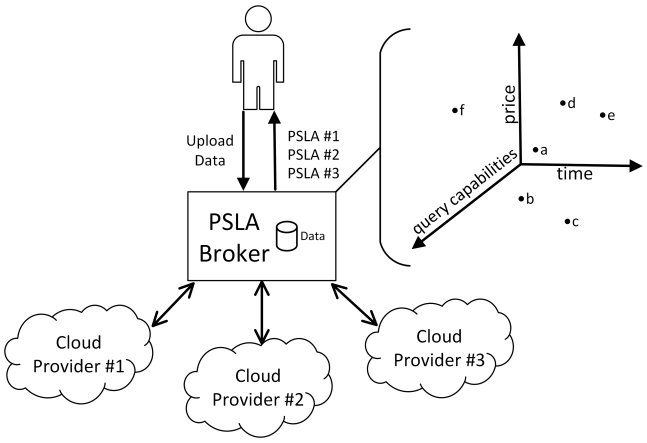


Figure 1: The PSLA system acts as a broker between different Cloud-provided data management services and the user. For each service, it generates alternative agreements with different trade-offs between cost, performance (query runtimes), and query capabilities. The user then selects the desired service and agreement.

As a second challenge, although Cloud providers offer availability through service level agreements (SLAs), they do not provide any type of performance guarantee. Studies suggest that there exist variances in the performance of jobs executed in Cloud services [15]. For instance, Schad et al. quantified the variance to be close to 20% in terms of CPU, I/O, and network performance through the Amazon Elastic Compute Cloud service [19].

In this paper, we lay out a vision for a radically new way for users to interact with their Cloud service providers. Instead of asking the user to specify the resources that she wants or asking the user for the exact queries she needs to execute, our proposed system looks at the user's data and shows the user what she can do with the data for a set price. The focus of our proposal is to ensure both high-performance and simplicity, two core features required by Cloud users [20]. Our vision is based on an innovative type of service level agreement between the user and provider as shown in Figure 1. The key idea is for a user to simply upload her data to a Cloud broker system and be presented with service-level alter-

natives that correspond to different points in the space trading off *query runtime performance*, *monetary cost*, and the *query capabilities* of the service. We call these agreements, personalized SLAs or PSLAs, because their detailed specification depends on the data uploaded by the user. Additionally, we strive for cost and performance predictability. Our PSLAs come with associated *fixed hourly service prices*, *guaranteed query performance*, and a clear specification of *query capability limitations*. The latter are shown in the form of templates for the types of queries that the user can execute. While prior work studied techniques that enable users to specify a desired price-performance curve when submitting a query [21], the approach simply rejects queries whose performance cannot be satisfied. Instead, we focus on informing users about what they can and cannot do with their data within various price and performance bounds. Figure 2 shows concrete PSLA examples for the Amazon Elastic MapReduce service and for Google BigQuery. We return to these examples later in the paper.

In the rest of the paper, we present the vision and challenges of offering PSLAs. As an initial step, we focus on PSLAs for structured data.

2. MOTIVATION

We begin with a motivating scenario to demonstrate the challenges with today's Cloud pricing models and SLAs.

2.1 Motivating Example

Sarah is a data scientist with access to a log capturing all the Web pages viewed by a set of users over some period of time. This scenario corresponds to the PigMix benchmark [18]. Sarah has access to 100GB of data but decides to explore a 10GB subset. The schema for the data is as follows:

Users (name, phone, address, city, state, zip)

PageViews(user, action, timespent, query_term, ip_addr, timestamp, estimated_revenue, page_info[1], page_links[1])

[1] These attributes are bags of maps

Sarah has the option to analyze her data using one of many existing Cloud services. She first turns to Amazon Web Services (AWS) [1], a popular Cloud provider today.

Sarah's first choice is between using Amazon Elastic MapReduce (EMR) or Amazon Relational Database Service (RDS). In both cases, she will be able to study her data by issuing declarative queries (Pig Latin or HiveQL in one case and SQL in the other). The challenge, however, is that the trade-off between price, performance, and capabilities for each service is not directly obvious from the online pricing models. Amazon RDS provides Sarah access to a commercial-grade, relational DBMS, but her analysis will be limited to a single node and the prices are higher: *e.g.*, \$0.17 to \$1.14 an hour for a SQL Server instance. Amazon EMR enables parallel processing and is cheaper with prices between \$0.06 and \$0.40 per instance and per hour [1], but Sarah will have to select the sizes and number of virtual machines that she needs.

This challenge is extended to the choice of Cloud providers. Each provider offers its own services for data analysis with a different pricing model and feature constraints. For example, Sarah could use Google BigQuery [5]. BigQuery is simple, as it allows a user to upload data and quickly execute SQL queries on the data. Currently, interactive queries are priced at \$0.035 per GB processed while batch queries are priced at \$0.02 per GB processed. Because BigQuery is column-based, the number of GB processed depends on the number of processed columns, which will be some subset of Sarah's 10GB, but estimating the exact query costs is not obvious. This pricing model thus still makes it difficult for Sarah to estimate the cost of analyzing her data and to compare against the cost of the analysis with the Amazon Cloud service.

Sarah decides to first try Amazon Elastic MapReduce and use Pig Latin to write her queries. Even with this concrete choice, Sarah still faces another set of decisions. Amazon's pricing model requires that she choose the number and sizes of instances in order to analyze her data. The choice will affect the price that Sarah will pay and the performance that she will get. The choice is thus important. Yet, no tool is available to help Sarah make such a decision, unless she defines a precise workload first [9].

Sarah decides to use the default configuration on Amazon EMR, that is, 1 small master node and 2 small slaves. She stores the data on the Amazon S3 storage and proceeds to spin up the cluster. The total cost adds up to \$0.18 per hour for the cluster and \$0.95 per month for storage. As she begins to run queries, she wonders if she can do better. Although her queries do return answers, she does not understand why some queries are faster than others. Would buying more machines improve the overall performance?

Similarly, Sarah decides to also try Google's BigQuery Cloud service. From the pricing scheme, it is unclear how many GB she will need to process in order to effectively explore her data. Thus, an accurate estimate for the end cost is nearly impossible. She will also need to keep in mind that there are certain limitations to BigQuery's SQL-like language. For example, operators such as DISTINCT are disallowed. Although BigQuery provides Sarah the ability to interactively query her data, there is no guarantee of how long it will take a query to return a response. Regardless, Sarah uploads her data and immediately has the option to query it. After running a few queries, she realizes that although some queries are fast, a self-join query on a 1GB subset of the data takes as long as 14 minutes. Although Sarah is willing to pay more for faster results, this option is not available through BigQuery.

This example illustrates that, today, it is difficult for users to select a Cloud provider and level of service that corresponds to a desired trade-off between cost, performance, and query capabilities. Users today need to proceed by trial and error when selecting service levels. Even for experts, finding cost-effective methods to process large datasets continues to be a nontrivial task [3].

2.2 Vision

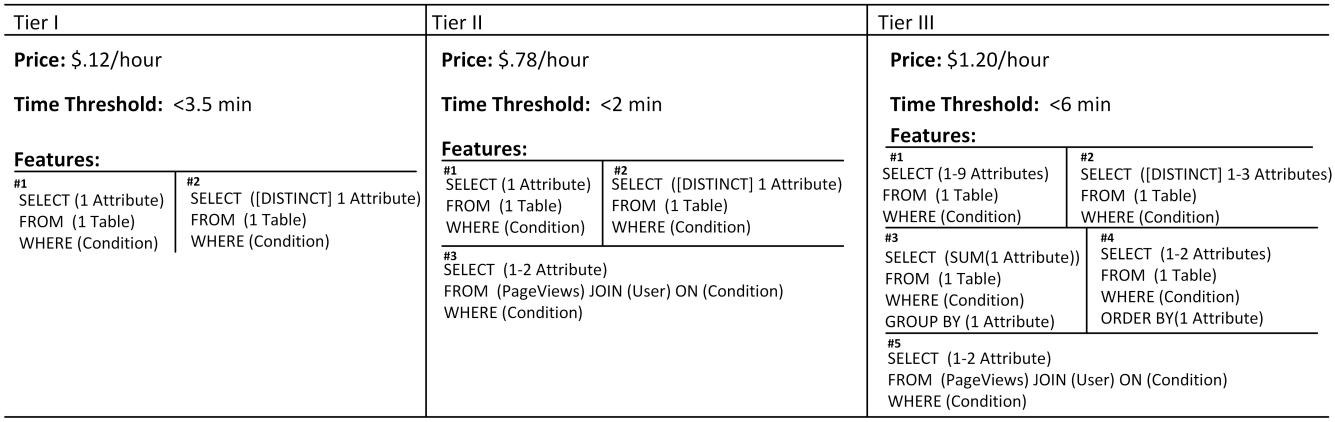
To address the above challenge, we propose to re-think the interface between users and Cloud services. Instead of forcing users to translate their price, performance, and capability goals into resource requests, we propose to automatically perform this translation for them. In particular, we want users to simply upload their data. The system should *automatically analyze that data* (*e.g.*, compute statistics on the data) and describe to users what they can and cannot do with their data based on price and performance factors. We call these descriptions, Personalized Service Level Agreements (PSLAs).

3. APPROACH

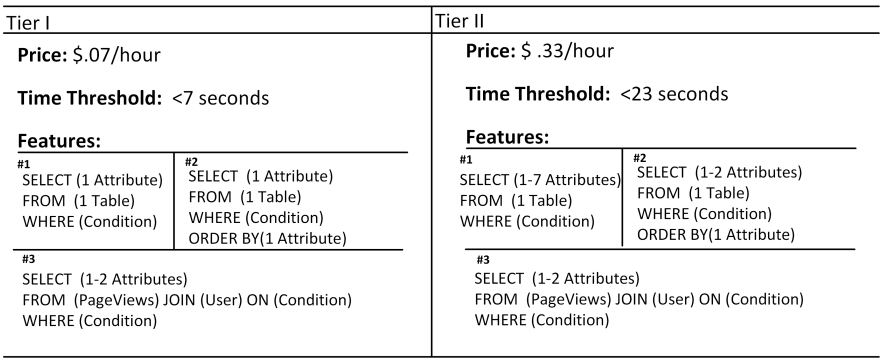
We first define a PSLA more precisely and give examples of manually derived PSLAs for a concrete use-case. We then discuss the challenges associated with generating such PSLAs automatically.

3.1 PSLAs

A PSLA is composed of a set of tiers $R_1, R_2 .... R_k$. Each tier



(a) PSLA for Amazon EMR



(b) PSLA for BigQuery

Figure 2: Example PSLAs for Amazon EMR and BigQuery.

corresponds to a unique level of service. That is, each tier offers a specific trade-off between query capabilities, price, and performance. No tier strictly subsumes another in the same PSLA. Each tier R_i, $1 \leq i \leq k$, has an hourly price P_i, a time threshold T_i, and a set of query templates $\{M_{i1}, M_{i2}, ..., M_{iv}\}$. Query templates define the capabilities available in the tier. The time threshold T_i guarantees that all queries which follow the templates will return within the specified time. The user selects one tier from the set. Specifically, a PSLA is a set of the form:

$$\begin{aligned} PSLA = \{&R_1 = (P_1, T_1, \{M_{11}, M_{12}, ..., M_{1v}\}),\\ &R_2 = (P_2, T_2, \{M_{21}, M_{22}, ..., M_{2v}\}),\\ &...,\\ &R_k = (P_k, T_k, \{M_{k1}, M_{k2}, ..., M_{kv}\})\} \end{aligned}$$

We illustrate the concept of a PSLA with a concrete use-case: We take a 10GB dataset from the PigMix benchmark [18] and manually generate PSLAs for two different Cloud data management systems: Amazon Elastic MapReduce (EMR) [1] and Google BigQuery [5].

To generate the query templates for the PSLAs, we consider a variety of relational operators including selection, grouping, aggregation, join, duplicate elimination, and sorting. We only consider simple queries that use a small number of these operators at the same time. We assume there are no indexes, and consider only explicitly specified primary and foreign key constraints.

In order to group query templates into an interesting set of tiers for Amazon EMR, we experiment with different numbers of instances, instance sizes, and Hadoop block sizes (the latter of which affects the degree of parallelism).

We generate three tiers for Amazon EMR. Each offers a different trade-off between query runtime, price, and query capabilities. Figure 2(a) shows the three tiers: Tier I offers users a cheap option for data analysis. It corresponds to a small configuration, but that detail is transparent to the user. In this tier, the PSLA promises the user only two types of templates they can run. We can also observe that this PSLA tier cannot promise any type of joins. Tier II offers a high-performance option with query runtimes under 2 minutes. The cost, however, is significantly higher. Finally, Tier III enables the largest variety of query templates, but it does so for an increased price and by relaxing the performance guarantee.

Overall, this example illustrates that one can easily derive meaningful service tiers that can help a user select a desired trade-off between price, performance, and capabilities. Most importantly, these service tiers enable high predictability: the user pays a fixed price, she can run a pre-defined set of queries (a query is never rejected if it follows the templates, unlike prior SLA proposals [21]), and the performance is guaranteed.

Using the same query templates and the PigMix dataset, we also manually derive a PSLA for Google BigQuery. PSLA generation is more difficult for this service for several reasons. First, its pricing model is based on GB processed. To derive a price per hour, we assume that a user will continuously execute the most expensive query in the tier. We also assume that the PSLA system can delay query results up to the promised runtime limit and we use this limit to determine how many queries can execute each hour. Second, while some queries in BigQuery execute within seconds, others, such as self-joins or queries that select an output for several columns were much slower. Finally, BigQuery does not support

query features such as duplicate elimination. Figure 2(b) displays the two tiers that we have derived. One tier is designed to be cheap while the other tier enables the use of a larger variety of queries.

Building queries on the 10GB dataset brought up interesting limitations as far as the type of operators that can execute efficiently in the Cloud. For example, a query that contained a self-join on the **Users** table was not able to finish under 2 hours on either system. On the other hand, for a 1GB subset of the data, the self-join took 14 minutes on BigQuery compared to 99 seconds on Amazon EMR. Our PSLAs can warn users about such queries by excluding them from the service tiers that cannot run them efficiently.

Through the PSLAs shown in Figure 2 comparing the performance, price and query capabilities of the two Cloud systems becomes significantly easier. BigQuery is inarguably faster, but it has stricter query limitations. Although the results are helpful, we cannot manually create a PSLA for each user and their data. The question is, how can we automatically create a PSLA based on the user's data and the characteristics of a Cloud provider?

3.2 Research Challenges

In this section we present some of the challenges behind PSLAs and provide insights as to how we plan to solve them. The key research challenges stem from the automatic generation of PSLAs.

System Architecture - We envision three possible deployments for the PSLA system. (1) Each Cloud provider can offer an integrated PSLA interface directly on top of its own data management services. This deployment enables the most accurate PSLAs, but requires that the user uploads her data to each cloud service separately and pays the associated storage costs. (2) An alternative option is to run the PSLA as a broker system as shown in Figure 1. The user first uploads her data to this broker, which then interacts with the various Cloud providers to generate one PSLA per Cloud. The user then chooses a desired PSLA. To support such a PSLA broker, Cloud providers can either (a) offer a PSLA interface based on statistics about data. The PSLA broker then only uploads the schema of the dataset and a set of locally collected statistics; or (b) offer no special PSLA-related functionality forcing the PSLA system to compute PSLAs using only the unmodified, existing Cloud interfaces. (3) Finally, the PSLA broker can run as a client-side application instead of a separate service. This approach removes the step of uploading data to the broker but burdens the user with installing the necessary software. In the rest of this paper, we assume the PSLA Broker deployment.

Query Template Selection - Given an input database comprising a set of relations, the number of queries that can be posed over the data is unbounded. The PSLA system, however, must show users a bounded set of query templates through each service tier. The key question is how to select *good* query templates. A small number of templates facilitates tier comparisons. However, keeping the number of templates small requires either (1) showing only templates for simple queries or (2) making the templates more generic.

Showing templates only for simple queries is the approach that we took for the PSLAs in Figure 2. Based on our prior experience working alongside domain scientists, we find that providing templates even for simple queries covers a significant amount of data processing needs. For example, the templates for Amazon EMR in the first tier correspond to selection queries, the following templates add joins, while the final set of templates add aggregation and sorting. Basic templates, however, may be insufficient for users with advanced data processing needs. The alternate approach of utilizing more generic templates also poses problems, though. In particular, generic templates reduce the number of possible tiers since some tiers are cheaper or faster because they give up certain query capabilities.

The key open questions associated with the selection of query templates are thus the following: (1) how many templates should suffice for each tier? (2) what is the trade-off between showing more precise templates that cover a smaller number of queries or broader templates that cover larger numbers of queries? (3) which queries should be covered by the templates and which queries can be omitted?

A promising approach that we are exploring is to refine query templates interactively. The system first shows tiers with simple templates as shown in Figure 2. If the query templates are not sufficient, the user can request to navigate stepwise into additional more complex templates. The detailed nature of this interactive template generation and the way the system enables user navigation, however, remain open problems.

Tier Selection - The input data can range in size from small (Gigabytes) to large (Terabytes), the set of possible query templates is unbounded, and the back-end Cloud service offers a large variety of configurations with different price/performance characteristics. Together, these three properties complicate the selection of service tiers. We observe, however, that each service tier is determined by three properties: (1) capabilities in the form of supported query templates, (2) performance in the form of runtime guarantee for queries that follow the templates in the tier, and (3) a price per hour. A good set of tiers should show alternatives at different points in this three-dimensional space.

The question then becomes which points to show? The approach that we advocate is to interactively refine this set of points, similarly to the interactive query template generation: For tiers, we propose to show users an initial set of points far apart in the 3D space. Then allow the user to "zoom in" on some regions of the space to see additional tiers in the subspace of interest. For example, a user may "zoom in" to see additional tiers that enable interactive data exploration (*i.e.*, all query times below 5 min) under $1/hour but giving up query features to a different extent.

The key research challenge with this approach is to develop algorithms that effectively and efficiently explore the 3D space of all possible tiers as the user zooms in or out in different regions of the space. In particular, the system should produce the skyline of best configurations at any time. By skyline, we refer to the operator that helps solve multicriteria decision making [16]. Given a set of PSLA tiers $R_1, R_2, ..., R_k$, the skyline returns all PSLA tiers R_i such that R_i is not dominated by another PSLA tier R_j.

Service Configuration Selection - When generating service tiers, another challenge is how to explore the set of possible cluster configurations and how to predict query runtimes in each setting. Accurate query runtime prediction remains an open problem, especially in multitenant settings [19]. The PSLA system must thus handle inaccurate query runtime estimates. Additionally, even with accurate estimates, queries expressed with the same template can have significantly different runtimes depending on template parameters and the resulting query plans. One approach to managing this variance is to raise runtime guarantees in the service tiers to account for prediction inaccuracies and expected runtime variances between queries for a single template. This ensures that all (or almost all) queries execute within the guaranteed time. The problem with this approach, however, is that a small number of outliers can skew the guarantees that the system can make rendering the proposed tiers useless. An alternate approach is to make guarantees probabilistic. For example a PSLA tier could guarantee that 90% of queries will execute under 1 min, 95% of queries will run under 5 min and 99% of queries will run under 10 min.

A question is whether the user could help the system decide how to handle outliers. A solution adopted by other research projects [21] is to reject queries that cannot execute within the guaranteed time and to compensate the user. However, a more useful approach might be to automatically scale resources, consequently increasing the costs for the user. Perhaps the user is willing to spend some limited amount of extra money in this fashion. The questions here are (1) When is it possible to detect inaccurate query runtime estimates? (2) How to scale resources in order to guarantee the performance threshold? (3) What if the system fails to meet the performance goals even after adding extra resources?

Cloud Heterogeneity and Evolution - Each Cloud service provider has specific *features* important to our context: price, performance and query capabilities. At the same time, some impose *constraints* on what is possible, restricting the possibilities in the 3D space of the PSLA (Figure 1). Furthermore, those providers are constantly *evolving*. The PSLA system should automatically leverage all the features and understand all constraints. It is important for this to be performed behind the scenes, which means the user does not need to be aware of Cloud service provider capabilities.

As examples from our experiments with Amazon EMR and Google BigQuery, we can cite as a feature that Amazon EMR allows one to tune the system by choosing the number and types of machines. As examples of constraints, Amazon EMR limits users to 20-node clusters and BigQuery does not support the keyword DISTINCT in queries. As an example of evolution (we captured during our experiments) BigQuery started to support joins between relations of any size by adding the EACH keyword right after JOIN; before that, it was only possible to use small right-side relations on the join (8MB of compressed data).

The research challenge here is how to automate the PSLA system in order to deal with all the features and constraints of different service providers. It is clear from the examples above that the features and constraints change the way the PSLA algorithm is able to generate all possible tiers in the 3D space. One possible solution to this problem is to create a formal declarative language to describe features and constraints of service providers. However, we still have open questions if we go further in this direction. Who is responsible to translate each Cloud service provider features and constraints into that language? How can we automatically detect and handle evolving features and constraints?

4. RELATED WORK

Significant research investigates how to build elastic DBMSs using Cloud infrastructures [4, 7, 8, 6]. Those systems, however, focus only on the overall performance of the DBMS, but not its SLAs. The ActiveSLA system [21] provides an admission control framework based on SLAs that tries to maximize profit given the SLA rules and rejects queries that cannot meet SLA objectives. In contrast, our PSLAs tell users what they can and cannot do with different price-performance choices. Hence, we never reject queries. The XCloud System [17] provides a formal language (grammar) for specifying users' performance needs by combining specific database metrics and operations. Our PSLAs hide system details from users. Zhao et al. [22] propose a framework to define and control database specific SLAs using a specific XML dialect. Their approach, however, focuses on OLTP workloads and their SLAs are then expressed in terms of replica freshness and transaction response times. Lang et al. [12] and Liu et. al. [14] study the problem of resource sharing across tenants in a multi-tenant environment and the impact on per-tenant SLA guarantees (either transactions per second or query latency). Multitenancy is a complementary problem to the one we present in this paper.

5. CONCLUSION AND OUTLOOK

We present a vision and associated research challenges for a new interface between users and data management Cloud services. Our core idea is to examine the concrete data that a user wants to process and to generate *a personalized service level agreement (PSLA)*, which shows templates for the types of queries the user can execute on her data for a given price and within a given runtime threshold. PSLAs thus enable users to focus on their main concerns of query performance, cost, and capabilities, freeing them from the current resource-centric approaches. The next step to enable this vision is to develop a method to generate good query templates and enumerate potential PSLA tiers given a specification of the capabilities of a Cloud service.

6. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation through NSF grant IIS-1247469, Petrobras, and gifts from EMC. J. Ortiz is supported by an NSF graduate research fellowship.

7. REFERENCES

[1] Amazon AWS. `http://aws.amazon.com/`.
[2] Azure. `http://www.windowsazure.com/en-us/`.
[3] S. Babu. Towards automatic optimization of MapReduce programs. In *Proc. of the First SoCC Conf.*, pages 137–142, 2010.
[4] P. A. Bernstein, I. Cseri, N. Dani, N. Ellis, A. Kalhan, G. Kakivaya, D. B. Lomet, R. Manne, L. Novik, and T. Talius. Adapting Microsoft SQL Server for cloud computing. In *Proc. of the 27th ICDE Conf.*, pages 1255–1263, 2011.
[5] Google BigQuery. `https://developers.google.com/bigquery/`.
[6] C. Curino, Y. Zhang, E. P. C. Jones, and S. Madden. Schism: a workload-driven approach to database replication and partitioning. *Proc. of the 36th VLDB Conf.*, 3(1):48–57, 2010.
[7] S. Das, D. Agrawal, and A. El Abbadi. ElasTraS: An elastic transactional data store in the cloud. *CoRR*, abs/1008.3751, 2010.
[8] S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi. Albatross: Lightweight elasticity in shared storage databases for the cloud using live data migration. *Proc. of the 37th VLDB Conf.*, 4(8):494–505, 2011.
[9] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proc. of the Second SoCC Conf.*, page 18, 2011.
[10] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Bridging the tenant-provider gap in cloud services. In *Proc. of the 3rd ACM Symp. on Cloud Computing*, pages 10:1–10:14, 2012.
[11] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The researcher's guide to the data deludge: Querying a scientific database in a just a few seconds. In *Proc. of the 37th VLDB Conf.*, volume 4, 2011.
[12] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan. Towards multi-tenant performance SLOs. In *Proc. of the 28th ICDE Conf.*, pages 702–713, 2012.
[13] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: comparing public cloud providers. In *Proc. of the ACM SIGCOMM Conf.*, pages 1–14, 2010.
[14] Z. Liu, H. Hacigümüs, H. J. Moon, Y. Chi, and W.-P. Hsiung. PMAX: Tenant placement in multitenant databases for profit maximization. In *Proc. of the EDBT Conf.*, pages 442–453, 2013.
[15] W. Lu, J. Jackson, J. Ekanayake, R. S. Barga, and N. Araujo. Performing large science experiments on Azure: Pitfalls and solutions. In *IEEE CloudCom*, CloudCom'10, pages 209–217, 2010.
[16] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM TODS*, 30(1):41–82, Mar. 2005.
[17] O. Papaemmanouil. Supporting extensible performance SLAs for cloud databases. *ICDEW*, 0:123–126, 2012.
[18] PigMix. Pigmix benchmark. https://cwiki.apache.org/PIG/pigmix.html, 2012.
[19] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc. of the 36th VLDB Conf.*, 3(1-2):460–471, Sept. 2010.
[20] E. Wu, S. Madden, Y. Zhang, E. Jones, and C. Curino. Relational cloud: The case for a database service. Technical Report MIT-CSAIL-TR-2010-014, CSAIL MIT, 2010.
[21] 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 *Proc. of the Second SoCC Conf.*, pages 15:1–15:14, 2011.
[22] L. Zhao, S. Sakr, and A. Liu. A framework for consumer-centric SLA management of cloud-hosted databases. *IEEE TSC*, page 1, 2013.