QUALITY OF SERVICE FOR DATABASE IN THE CLOUD

Flávio R. C. Sousa, Leonardo O. Moreira, Gustavo A. C. Santos and Javam C. Machado
Departament of Computer Science, Federal University of Ceara, Fortaleza, Brazil

Keywords: Cloud Computing, Data Management and Quality of Service.

Abstract: Cloud computing is a recent trend of technology that aims to provide on-demand services following a pay-per-use model. In the cloud, the service user has some guarantees, such as performance and availability. These guarantees of quality of service are defined between the service provider and user and are expressed through a service level agreement. There are many models for agreement and quality of services in cloud computing. However, most of these models are multipurpose and do not deal with data management aspects in the cloud. This paper presents QoSDBC, an approach to quality of service for database in the cloud. This approach can be used by providers to improve the quality of their services and encompasses different aspects such as response time, throughput, availability and consistency. In order to evaluate QoSDBC, some experiments that measure the quality of service are presented.

1 INTRODUCTION

Cloud computing is an extremely successful paradigm of service-oriented computing. Scalability, elasticity, pay-per-use pricing, and economy of scale are the major reasons for this success. Since the majority of cloud applications are data-driven, database management systems (DBMSs) powering these applications are critical components in the cloud software stack (Elmore et al., 2011). Many companies expect cloud providers to guarantee quality of service (QoS) using service level agreements (SLAs) based on performance aspects. Nevertheless, in general, providers base their SLAs only on the availability of services. Therefore, it is crucial that providers offer SLAs based on performance for their customers (Schad et al., 2010). For many systems, most of the time consumed on service provision is related to the DBMS, rather than on front-end web/app server. Thus, data management service must be monitored in order to assure the expected performance and availability on cloud system service (Schroeder et al., 2006).

There are many models for SLA and QoS in cloud computing (Fito et al., 2010), (Malkowski et al., 2010) (Schnjakin et al., 2010) (Mazzucco, 2010) (Ferretti et al., 2010) (Wu et al., 2011). Those models are multipurpose and do not deal with data management aspects. There are also specific models for SLAs and quality of database service that provide solutions in this context (LSCR, 2011) (Yang et al., 2009) (Xiong et al., 2011) (Chi et al., 2011). Nevertheless, these models fail to address some aspects of data management, such as service-specific metrics of databases, and provide only part of a solution for QoS (e.g. the SLA definition or approaches to penalties). Furthermore, these works do not use specific techniques for monitoring DBMSs. According to our research, there are no solutions that address this problem, since previous works have focused only on some of its aspects.

In order to solve this problem, this paper proposes QoSDBC [1], an approach to QoS for database in the cloud. This approach can be used by providers to improve the quality of their database services and encompasses different aspects such as response time, throughput, availability, and consistency. The major contributions of this paper are (i) an approach to support QoS for database in the cloud (ii) a SLA specification for database in the cloud (iii) the definition of a set of metrics for monitoring the database service (iv) the implementation of the proposed approach and its architecture, and finally (iv) an evaluation of the approach and results that show its efficiency. This paper is organized as follows: in section 2, QoSDBC theoretical aspects are presented. The implementation of the solution is described in section 3. The evaluation of QoSDBC is presented in section 4. Section 5 details related work and, finally, section 6 presents the conclusions.

[1] *Q*uality of *S*ervice for *D*ata*B*ase in the *C*loud.

2 QoSDBC

The QoSDBC is a solution to provide database quality of service on cloud platforms. It addresses various issues such as SLA definition, monitoring techniques, and penalties. There are many solutions to improve QoS, such as cache, scheduling, adaptive query processing, capacity provisioning, and replication. In this paper, we employ database replication to ensure QoS, which is appropriate to improve availability, performance, and scalability in different environments (Özsu and Valduriez, 2011). QoSDBC provides database services in the Platform as a Service (PaaS) model that can easily take advantage of cloud based infrastructure. It simplifies the use of database in the cloud and can be integrated with existing cloud infrastructures. Solutions for QoS in the cloud may be classified as cloud provider centric or customer-centric. Cloud provider centric approaches attempt to maximize revenue while meeting an application's SLA in the face of fluctuating workloads. In this work, we focus on the cloud provider centric approach.

In cloud systems, SLAs have different purposes, but it is possible to identify a general structure for them: information about the parties, SLA parameters, metrics used to calculate the SLA parameters, algorithms to calculate the SLA parameters, service level objective (SLO) and actions to be performed in case of agreement violation (Schnjakin et al., 2010). In this paper, we propose the following definition:

Definition. *A SLA for database service in the cloud is composed of information from the parties, SLA metrics, SLOs, algorithms to calculate metrics, and SLA penalties.*

Information about the parties refers to the contract between the provider and the customer. SLA metrics are related to the items to be monitored, like response time and throughput, while SLO contains pre-defined limits for the parameter, as response time lower than 5 ms. There is a way to calculate it (e.g. average) as well as penalties in case of non-compliant SLOs (e.g. fine). According to (Chi et al., 2011), SLA metrics for cloud databases should optimize the system, address relevant issues for data management and consider the specificities of cloud computing model. QoSDBC makes use of metrics of response time, throughput, availability, and consistency. A SLO is associated with each metric, as follows.

- *Response time*: the maximum response time, in seconds, for each query.
- *Throughput*: minimum output, in transactions per second.
- *Availability*: maximum fraction of rejected queries over a period of time *t*.
- *Consistency:* access to updated data according to the consistency type can be strong or weak.

In QoSDBC, the SLA is profit-oriented. The profit-oriented SLA presents a reliable operation of the systems, since the provider is motivated to provide a high-quality service. Revenue is the amount paid by the customer to the provider to meet an SLA S_i, and operating cost can be defined as the expense of the provider to perform a service with a specified SLA S_i. Thus, profit is the sum of all corresponding receipts minus operating cost plus the sum of all penalties, as showed in the following formula.

$$Profit = Revenue - (Cost + Penalties) \tag{1}$$

Penalty is an amount that the provider must pay to the customers if the SLA S_i is not met. For example, in Google AppEngine, Microsoft Azure, or Amazon S3, if availability is lower than 99.9%, then, the customers receive a service credit, according to SLA, and proportional to the revenue. Similarly, the response time is critical to ensure QoS and may incur in penalties in some service models (Xiong et al., 2011). In QoSDBC, we define the penalty cost as the ratio of the sum of all queries that did not meet SLOs to the total queries multiplied by the system revenues, according to the formula below.

$$Penalties = \frac{\sum Violated\_Query}{\sum Query} * Revenue \tag{2}$$

Consequently, we can define a satisfaction function for the SLA, as showed below. The function is satisfied if the SLA S_i is satisfied, i.e., all SLOs from the SLAs S_i are satisfied; otherwise, it is not.

$$FSS(Si) = \begin{cases} 1 & \text{if SLA Si satisfied} \\ 0 & \text{if SLA Si not satisfied} \end{cases} \tag{3}$$

2.1 Monitoring of SLA Metrics

Response time is usually used to check the QoS. However, in many contexts, it is important to establish more general goals for QoS (Schroeder et al., 2006). The percentile is requested by users as part of an SLA, for example, to ensure that at least 90% of customer transactions have a response time below a specified limit (Entrialgo et al., 2011). For each metric of the SLA, you can use an algorithm to compute the SLA metrics. QoSDBC makes use of the following strategy:

- *Response time*: x% percentile of response times which are lower than a value *y* during a period of time *t*.

- *Throughput*: z% percentile throughput greater than a value *k* during a period of time *t*.
- *Availability*: function satisfied/not satisfied according to the formula MTTF/(MTTF + MTTR), where MTTF - Mean Time Between Fail and MTTR - Mean Time To Repair.
- *Consistency*: function satisfied/not satisfied.

QoSDBC makes use of the time interval of one hour to check the SLA penalties, since this value is used by most providers to charge resources. In order to set monitoring boundaries, we propose the following SLA states, as shown in Figure 1:

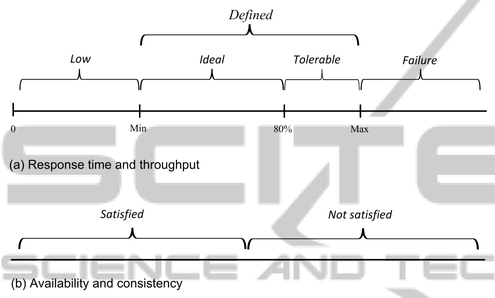
Defined
Low
Ideal
Tolerable
Failure
0
Min
80%
Max
(a) Response time and throughput
Satisfied
Not satisfied
(b) Availability and consistency

Figure 1: States of SLA.

- *Low*: The SLA is lower than that set by the customer. In this state, resources can be removed from the system.
- *Defined*: The defined level is divided into ideal and tolerable. In the ideal range, the SLA is maintained within an acceptable range. In the tolerable range, the system intensifies monitoring in order to define the addition of resources.
- *Failure*: At this level, a failure occurred in relation to the SLA. In this case, the provider is penalized according to the number of queries at the failure level and new resources must be added quickly to return to the defined level.

Due to their representativeness, response time and throughput are high-level performance metrics that need to be collected and analyzed. The values of these metrics depends on the state of the database metrics. When the database system is not overloaded, the values are almost constant. However, when it is overloaded, the values grow linearly and then exponentially. Thus, it is necessary to have effective mechanisms to detect the increase or decrease of these values (Schroeder et al., 2006).

QoSDBC uses an efficient strategy to calculate data collected and combines different monitoring techniques to treat the variability of metrics. The collect process is performed six times with an interval of 10 seconds. For each collect process, QoSDBC calculates the median and standard deviation. Two medians with lower deviation are selected as the final values to be stored. To these values, it is applied an exponentially weighted moving average X'_t = α X_t + (1 - α) X'_{t-1}. This monitoring technique is based on the work (Fito et al., 2010).

3 SYSTEM ARCHITECTURE AND IMPLEMENTATION

QoSDBC architecture is divided into two parts: Agent and QoSDBCCoordinator. The QoSDBCCoordinator consists of a set of services that address the management of resources. The Agent is a component added to each VM which is responsible for interacting with the VM and the DBMS. Specifically, this agent monitors and interacts with the DBMS, while checking the state of monitored resources. An overview of the QoSDBC architecture is shown in Figure 2.

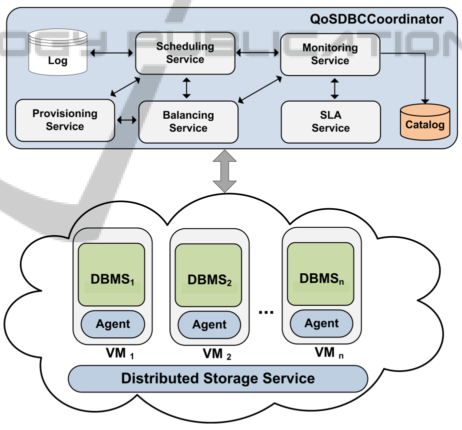

Figure 2: QoSDBC architecture.

The Monitoring Service is responsible for managing the information about the state of the VMs and the DBMS collected by the agent. For each DBMS, CPU resources, memory and databases sizes, as well as SLA metrics, are monitored. The SLA Service manages agreements between costumers and service provider and the Balancing Service distributes the requests to the resources. The Provisioning Service defines the resources required to ensure QoS. A catalog stores the data collected and information about the resources. Finally, the Scheduling Service directs requests to the VMs and keeps a log of the last transactions submitted to the system. The data is persisted in a distributed storage service. QoSDBC implements a driver to encapsulate the services and a complete abstraction for the developer over the architecture. The SLA metrics are calculated directly at the service provider, since it would be more complex to perform measurements on the customers due to variations in the connection quality.

We have implemented a prototype of QoSDBC in Java. We assume a virtualized cloud platform and, each virtual machine (VM) image is pre-configured with the agent; thus, when new VMs are dynamically deployed, the QoSDBC automatically recognizes new servers and begins to monitor them without the need for any additional configuration. The agent collects information about the database. Resource metrics can be obtained directly by query through the APIs Amazon Elastic Compute Cloud (EC2) CloudWatch. The monitoring data is stored in an Amazon SimpleDB database. We use the Java APIs exposed by Amazon EC2 to deploy, terminate, or reconfigure servers allocated to a database service. QoSDBC makes use of a simplified version of the WSLA language (Keller and Ludwig, 2003) to deal with the SLAs management.

QoSDBC implements elasticity by adding and removing replicas according to the workload. To add a new replica, a new VM is started and the new replica is added in this machine. The new replicas are added and updated through data migration. Data migration involves applying snapshot and logs of missing updates to the new replica to bring it up-to-date. For the migration process, we are using XtraBackup backup tool (**?**), which has high performance and allows hot backup. With reduction in the workload, QoSDBC removes replicas of the database. In order to accomplish this, QoSDBC selects the replica with the lowest workload and stops sending requests to it.

4 EVALUATION

The evaluation of cloud database services presents significant differences when compared to the evaluation of non cloud systems. In the cloud environment, the goal is to minimize or to adjust the amount of resources needed to ensure QoS. The wide range of cloud database services and how these systems are built (e.g. data model, levels of consistency, query language) turns difficult the development of a standard benchmark (Elmore et al., 2011).

4.1 Environment

For the experiments, small instance VM at Amazon's EC2 were used. Each machine runs the Ubuntu 11.10 operating system and MySQL 5.5 DMBS. We developed a benchmark in order to generate the workload to be executed by QoSDBC. This benchmark was developed based on BenchmarkSQL (BSQL, 2011), a Java implementation of TPC-C (TPC, 2011). Our benchmark allows addition and removal of customers at runtime, which allows us to evaluate the QoS with different workloads. It allows generating transactions in accordance with the parameters set, sending them to QoSDBC and collecting the results at the end of each execution. Transaction concurrency is simulated using multiple customers.

4.2 Experiments

We evaluate QoSDBC's quality of service by comparing it with static provisioning strategy in two cases: under-provisioning and over-provisioning. The static provisioning configuration uses a constant set of replicas for each service, that is 2 replicas in under-provisioning and 6 replicas in over-provisioning. For the QoSDBC approach, two replicas were initially used and then their amount changes according to workload. In this case, we used a VM with a full replica of the database. Thus, it was possible to divide the workload between the machines and to ensure quality. In this evaluation, the primary copy protocol was used. Other protocols such as Paxos commit (Özsu and Valduriez, 2011), can also be used. The following values of SLA parameters were defined: response time less than 0.5 seconds, 99.9% of availability, strong consistency, and the percentile response time by 95%. As the response time was used as a performance metric, we chose not to use the throughput in this experiment. According to TPC-C, a database of approximately 1 GB was generated.

To analyze the QoSDBC execution, an experiment was conducted varying the number of customers over a period of time. The experiment consists of running the system with different number of customers every 20 minutes, as shown in the following ordered pairs, which represent (time in minute, number of costumers): (20,30), (40,40), (60,60), (80,60), (100,80), (120,80), (140,60), (160,40), (180,30), (200,30), (220,20), and (240,20). The interval between the addition and removal of customers is similar to (Cecchet et al., 2011). On a public cloud such as Amazon AWS, the cost can be defined by the price the user is going to pay for the compute hours of the instance, the I/Os on Elastic Block Storage (EBS) and the monthly cost for data storage. Since the main cost is related to the instances, in these experiments we consider only this cost.

Figure 3 shows the variation of SLA response time metric with under-provisioning and QoSDBC. Initially, the SLA keeps the defined state with under-provisioning. The SLA response time increases after the first hour, because new customers were added at this time. With the addition of more customers, the SLA switches to failure state, resulting in penalties to the service provider until reducing the amount of customers, which occurs after 160 minutes. With the decrease in workload, the strategy with static provisioning does not reduce the amount of resources, leading to constant costs. With QoSDBC, the SLA remains at the defined state. With the addition of customers, SLA response time is increased and the SLA goes to the failure state. However, QoSDBC detects this change and adds two new replicas of the database, which can handle part of the customers' requests. Thus, the system returns to the defined state, avoiding new penalties. With the decrease in workload, QoSDBC removes two replicas and reduces costs, although still maintaining the QoS.

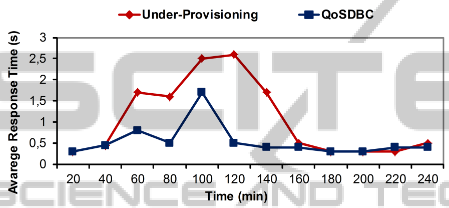

Figure 3: Avarege Response Time SLA with Under-Provisioning and QoSDBC.

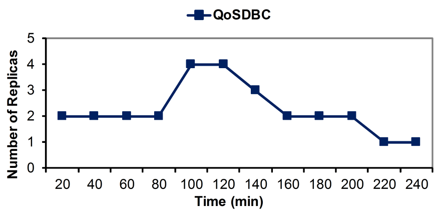

Figure 4: Replica allocation.

Figure 4 shows the replica allocation with changing load. The QoSDBC used a variable number of replicas according to workload. For this reason, its VM cost was slightly higher than under-provisioning, since this strategy uses two replicas throughout the whole experiment. Figure 5 presents the SLA violation, where 1 means satisfied and 0 means non-satisfied according to the satisfaction function for the SLA. The response time of the static provisioning grows rapidly with the addition of more customers. This happens because each replica has a limit on the amount of queries it can process and manage. Therefore, more than 50% of the queries was not answered in accordance with the SLA. For QoSDBC, less than 28% of the queries was not answered in accordance to the SLA. This occurred due to the time required to add a new replica and to migrate the data.

As the amount of revenue is greater than the costs with infrastructure resources (e.g. VMs), and the penalties are applied considering the revenue, the

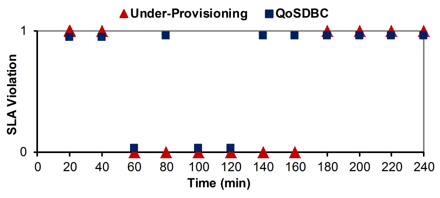

Figure 5: SLA violation.

under-provisioning approach reaches a high value in the penalties. So, the value of the provider's profit with the under-provisioning is lower than using the QoSDBC approach. Over-provisioning (i.e. 6 replicas) always provides an adequate capacity but at a significantly larger cost. In contrast, QoSDBC uses much less resources while still providing the required SLA. Furthermore, QoSDBC ensures the QoS, maintaining customer satisfaction. In these experiments, the availability and consistency metrics were met.

5 RELATED WORK

In (Yang et al., 2009), it is presented a platform that uses the definition of SLA for database considering throughput and availability. However, this work does not show how to calculate the SLA metrics neither the penalties for failure. (Balazinska et al., 2011) discuss different pricing models for cloud data services, but does not present SLA definitions. In (Zhu and Zhou, 2011), it is presented an approach to scheduling in order to provide QoS. In addition, it also presents a quality model and SLA for database systems using the model key/value, including penalties. That work focuses on scheduling policies. It does not address profit issues and can only be used in approaches that use key/value.

(Chi et al., 2011) propose a framework to support decision-making oriented to profit. This framework uses a new data structure called SLA-tree in order to support decisions in database services in the cloud, such as scheduling and capacity planning. The SLA is defined for each system query. Using SLA for queries makes its use complex, because the user needs to know in advance all queries to define the SLA. Moreover, this approach uses only response time as metric and does not consider penalties. In (Xiong et al., 2011) SmartSLA is presented, a system of intelligent resource management that takes into account aspects of cost, workload and cost of infrastructure. That paper describes an SLA for queries and penalties for failure. Similarly to (Chi et al., 2011), considering SLA for queries turns its usage complex and the only considered metric is response time. Amazon Auto Scaling (Amazon, 2011a) allows consumers to scale up or down according to criteria such as average CPU utilization across a group of compute instances (e.g. Amazon Relational Database Service (Amazon, 2011b)). Nevertheless, it does not take into account the database service state such as query response time or throughput. In addition, it uses only resource oriented metrics and does not implement SLAs to define the QoS.

6 CONCLUSIONS AND FUTURE WORK

This work presented QoSDBC, an approach to quality of service for database in the cloud. We evaluated the QoSDBC approach considering quality of service characteristics. According to the analysis of the obtained results, we found that the QoSDBC includes the characteristics of a database service in the cloud and can be used by providers to improve the quality of their services. As future work, we intend to conduct further experiments considering new scenarios and costs to better evaluate the QoSDBC. Other important issues to be addressed are related to new strategies for monitoring, penalties, and other aspects to be added to the SLA. Finally, we intend to conduct a study with techniques of machine learning to improve resource management and to add support to multi-tenant models.

ACKNOWLEDGEMENTS

This work is partly supported by Amazon AWS Research Grant.

REFERENCES

Amazon (2011a). *Amazon Auto Scaling*. http://aws.amazon.com/autoscaling/ .

Amazon (2011b). *Amazon Relational Database Service (RDS)*. http://aws.amazon.com/rds/.

Balazinska, M., Howe, B., , and Suciu, D. (2011). Data markets in the cloud: An opportunity for the database community. *PVLDB*, 4(12):1482–1485.

BSQL (2011). *BenchmarkSQL*. http://www.sourceforge.net/projects/benchmarksql.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25:599–616.

Cecchet, E., Singh, R., Sharma, U., and Shenoy, P. (2011). Dolly: virtualization-driven database provisioning for the cloud. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, pages 51–62, New York, NY, USA. ACM.

Chi, Y., Moon, H. J., Hacigümüş, H., and Tatemura, J. (2011). Sla-tree: a framework for efficiently supporting sla-based decisions in cloud computing. In *EDBT '11*, pages 129–140, New York, NY, USA. ACM.

Das, S., Nishimura, S., Agrawal, D., and El Abbadi, A. (2011). Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration. *Proc. VLDB Endow.*, 4:494–505.

Elmore, A. J., Das, S., Agrawal, D., and El Abbadi, A. (2011). Zephyr: live migration in shared nothing databases for elastic cloud platforms. In *SIGMOD '11*, pages 301–312.

Entrialgo, J., García, D. F., García, J., García, M., Valledor, P., and Obaidat, M. S. (2011). Dynamic adaptation of response-time models for qos management in autonomic systems. *J. Syst. Softw.*, 84:810–820.

Fito, J. O., Presa, I. G., and Guitart, J. (2010). Sla-driven elastic cloud hosting provider. *PDP, Euromicro'10*, 0:111–118.

Keller, A. and Ludwig, H. (2003). The wsla framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, 11:57–81.

Kossmann, D., Kraska, T., and Loesing, S. (2010). An evaluation of alternative architectures for transaction processing in the cloud. In *SIGMOD '10*, pages 579–590. ACM.

LSCR (2011). *SLA for database projects*. http://lscr.berkeley.edu/rates/sla/database.php.

Malkowski, S., Hedwig, M., Jayasinghe, D., Pu, C., and Neumann, D. (2010). Cloudxplor: a tool for configuration planning in clouds based on empirical data. In *SAC '10*, pages 391–398, New York, NY, USA. ACM.

Mazzucco, M. (2010). Towards autonomic service provisioning systems. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 273–282, Washington, DC, USA. IEEE Computer Society.

Özsu, M. T. and Valduriez, P. (2011). *Principles of Distributed Database Systems, 3rd Edition*. Springer.

Schad, J., Dittrich, J., and Quiané-Ruiz, J.-A. (2010). Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB*, 3(1):460–471.

Schnjakin, M., Alnemr, R., and Meinel, C. (2010). Contract-based cloud architecture. In *CloudDB '10*, pages 33–40, New York, NY, USA. ACM.

Schroeder, B., Harchol-Balter, M., Iyengar, A., and Nahum, E. (2006). Achieving class-based qos for transactional workloads. In *ICDE '06*, pages 153–, Washington, DC, USA. IEEE Computer Society.

Sharma, U., Shenoy, P., Sahu, S., and Shaikh, A. (2011). A cost-aware elasticity provisioning system for the cloud. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ICDCS ’11, pages 559–570, Washington, DC, USA. IEEE Computer Society.

TPC (2011). *Transaction Processing Performance Council*. http://www.tpc.org/.

Welsh, M. and Culler, D. (2003). Adaptive overload control for busy internet servers. In *USENIX USITS*, pages 4–4.

Wu, L., Garg, S., and Buyya, R. (2011). Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195 –204.

Xiong, P., Chi, Y., Zhu, S., Moon, H. J., Pu, C., and Hacigumus, H. (2011). Intelligent management of virtualized resources for database systems in cloud environment. In *International Conference on Data Engineering*, volume 0, pages 87–98, Los Alamitos, CA, USA. IEEE Computer Society.

Yang, F., Shanmugasundaram, J., and Yerneni, R. (2009). A scalable data platform for a large number of small applications. In *CIDR*, pages 1–10.

Zhu, Y., S.-M. and Zhou, X. (2011). Scheduling with freshness and performance guarantees for web applications in the cloud. In *Australasian Database Conference (ADC 2011)*, volume 115, pages 133–142, Perth, Australia. ACS.