Quantifying the Impact of Replication on the Quality-of-Service in Cloud Databases

Rasha Osman
Faculty of Mathematical Sciences
University of Khartoum
Khartoum, Sudan
Email: rosman@ieee.org

Juan F. Pérez
School of Mathematics and Statistics
University of Melbourne
Melbourne, Australia
Email: juan.perez@unimelb.edu.au

Giuliano Casale
Department of Computing
Imperial College London
London, United Kingdom
Email: g.casale@imperial.ac.uk

***Abstract*—Cloud databases achieve high availability by automatically replicating data on multiple nodes. However, the overhead caused by the replication process can lead to an increase in the mean and variance of transaction response times, causing unforeseen impacts on the offered quality-of-service (QoS). In this paper, we propose a measurement-driven methodology to predict the impact of replication on Database-as-a-Service (DBaaS) environments. Our methodology uses operational data to parameterize a closed queueing network model of the database cluster together with a Markov model that abstracts the dynamic replication process. Experiments on Amazon RDS show that our methodology predicts response time mean and percentiles with errors of just 1% and 15% respectively, and under operational conditions that are significantly different from the ones used for model parameterization. We show that our modeling approach surpasses standard modeling methods and illustrate the applicability of our methodology for automated DBaaS provisioning.**

I. Introduction

Cloud based applications and services must provide low latency and high availability, as increased response times have been shown to impact the revenue of large service providers [1]–[3]. In modern data-intensive applications, low latency at the application level depends heavily on the performance of the database (DB) layer. Database administrators (DBAs) must manage, upgrade and troubleshoot backend cloud databases in order to meet stringent service level agreements (SLAs) that are measured at above the 90^{th} percentile [4]. To alleviate such tasks, major vendors offer DBaaS (Database-as-a-Service) platforms such as Amazon Relational Database Services (RDS) [5], Microsoft Azure [6], Google Cloud SQL [7] and Oracle Database Cloud Service [8]. These services provide cloud-hosted database instances with semi-automated administrative and monitoring services.

Several methodologies have been proposed to assess quality-of-service (QoS) in component-based enterprise applications [9], however when evaluating the data layer these methods oversimplify the model of the database [10] or ignore the specific features of DBaaS offerings. In this paper we address this limitation by proposing an analytical method to analyze QoS in DBaaS architectures. Specifically, we present a methodological framework that enables the evaluation of the impact of replication on DBaaS platforms. Our methodology relies on queueing network theory and therefore can be integrated within existing methodologies that exploit these models for QoS assessment, e.g., [11]–[15].

DBaaS providers offer two types of database replication. Replication for failover and disaster recovery, in which, a *standby* database is synchronously updated from the primary database and serves as a backup. In addition, Amazon RDS and recently Google SQL (beta version), provide replication for *performance*, in which read workloads are offloaded from the primary database to read-only replicas that are updated asynchronously. This offers the DBA greater flexibility at the price of limited control and reduced features, e.g., Amazon RDS limits the number of replicas per primary database to five only [16]. In the rest of this paper, we will use the term *replication* to refer to replication for performance.

For cloud applications, improved performance and high availability are achieved through expanding the customer-facing services by increasing the number of resources where the application is deployed, known as horizontal scaling. For cloud databases, this is achieved by replicating the database on extra nodes in order to sustain the offered QoS during periods of high demand. The ability of the system to execute runtime horizontal scaling is known as *elasticity* [17], [18]. In the database context, elasticity refers to the ability to expand/retract the database instances with minimal delay and performance impact [17]. It is characterized by [18] (i) the time to *stability*, i.e., resuming normal performance, and (ii) the impact on performance during expansion/retraction.

Benchmarking studies [17]–[19] have shown that replication of cloud databases leads to performance instability, which depends on (a) the addition of a new node to the cluster, (b) the time to redistribute the data to the new node and (c) the speed in which nodes are added/removed from the cluster. The performance impact depends both on the underlying system resources and on the amount of data to be redistributed within the cluster [17]. Therefore, DBAs must tradeoff the benefit of replication against the possible SLA violations during the replica creation phase. As a result, evaluating the impact of replication on the database QoS can guide DBAs in decisions about when to scale the database cluster in and out, and with how many replicas, in order to guarantee SLAs.

In the performance modeling area, studies have concentrated on traditional relational databases using analytical models, e.g, [10], [20] or simulation-based techniques, e.g. [21], [22]. For distributed and replicated databases, the main issue considered by modeling studies is to accurately represent the routing of queries and communication between different nodes within the replicated database [23]. Elnikety [24] evaluates

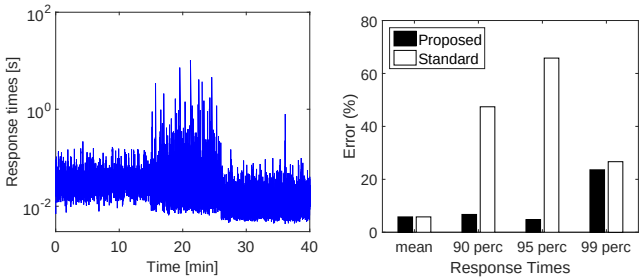
the performance of a replicated database by parameterizing an analytical model by measuring the performance of a standalone database. These studies investigated the overall mean response times and throughputs of replicated databases but do not consider percentiles, which are essential for DBaaS SLA guarantees. In addition, they do not explicitly investigate the impact of replication and replica creation and integration on the performance of replicated databases.

In this paper, we introduce a methodological framework that enables the evaluation of the impact that replication has on QoS for DBaaS platforms. We propose models that explicitly consider the variable nature of request processing times and the different system configurations that the DB cluster undergoes. We show that minimal non-intrusive measurements are enough to parameterize the model. Further, we are able to determine the response-time percentiles for the different phases of replication and overall, such that expected violations of SLAs can be identified and alleviated. Our modeling results are compared to a MySQL replicated cluster hosted on Amazon RDS and to trace-driven simulation.

Currently there is an increased interest in developing cloud provisioning tools that automate the deployment by using optimization techniques that rely on underlying modeling formalisms, e.g., SASSY [25], PerOpteryx [26], Skymark [27] and SPACE4Clouds [28]. Optimal provisioning decisions based on cost, SLA guarantees and workload fluctuations are guided by performance models that evaluate the expected performance of each scenario. We anticipate that the models produced from our methodology can be easily plugged into these tools, extending their applicability to DBaaS environments.

The main contributions of this paper are:

- We exploit fluid modeling techniques to approximate response time percentiles for replicated relational DBaaS platforms. Our models incorporate the different replication phases by means of an *environment model* [29], which captures the changes in DBaaS service times due to replica creation and data migration.
- We provide a simple methodology based on client-side measurements to evaluate the performance of a relational database cluster hosted on DBaaS platforms. This approach overcomes the limitations of the restricted access that such platforms provide (see Section II-A).
- We show that our modeling approach is able to predict the response time mean and *percentiles*, which are key in evaluating SLAs, for the overall performance of a production-level database cluster, as well as for each transient stage during database replication.
- We evaluate our methodology under variable workloads and dynamic cluster re-configurations, illustrating its ability to represent the intrinsic performance characteristics of DBaaS environments.
- We evaluate the impact of replication speed and overhead on performance stability and its effect on DB cluster performance, as measured by the response time percentiles. In addition, we illustrate the applicability of our methodology in guiding replication scale-out decisions in DBaaS environments.



(a) Impact on response times (b) Prediction errors

Fig. 1: The impact of replication on query response time.

The rest of this paper is organized as follows. Section II details the motivation behind this work and Section III describes our methodology for DBaaS evaluation. The models used to evaluate the replicated database cluster are described in Section IV. We describe the experimental setup for our running case study in Section V. Our methodology is validated against measurements in Section VI. Scaled workloads and dynamic replication are investigated in Section VII. Trade-offs in DBaaS scenarios are studied in Section VIII. Conclusions and future work are discussed in Section IX.

II. Motivating Example

DBaaS platforms offer high flexibility and a pay-as-you-go pricing model that is attractive to DBAs, especially for applications that face varying workloads. An example is the Amazon Relational Database Service (RDS) [5], which provides database administrative services, such as creation, loading, and replication through publicly available APIs. In addition, it supports automated *read* replica creation from a source database instance hosted on Amazon RDS. The source database is the *master* that serves read and write requests, while the replica (slave) can only serve read requests. During replica creation, Amazon RDS takes a snapshot of the source database (the master), which is used to create a replica node that is kept up-to-date asynchronously without user intervention [16]. In the rest of this paper, we refer to read replicas simply as *replica*.

During the replica creation and data migration process the performance of queries on the database can be affected by the backup and migration operations, potentially leading to SLA violations. For example, Amazon RDS estimates a one minute I/O suspension on the master DB as the database snapshot is created for replication [16]. Further, DBaaS offerings have been shown to exhibit high variability in query response times for single-instance relational databases [30]. For DBAs, this translates to a trade-off between adding a new replica to achieve better performance and the probability of SLA violations during replica creation.

To illustrate this impact, using Amazon RDS (see environment description in Section V), we ran queries against the master database and then initiated a replica creation from the master DB. Figure 1(a) shows the actual query response times before, during, and after replica creation. During the replica creation phase, the mean query response time increases by over 20% from $0.035s$ to $0.044s$ and the standard deviation of the response time increases almost 10 fold from $0.018s$

to $0.14s$. Such scenarios are difficult for DBAs to evaluate, especially as the degradation in performance happens at a time when the overall system is experiencing a high load, which is in fact the reason to expand the database cluster. In such scenarios, standard queueing modeling approaches [31] fail to represent the variability of the database performance, especially when dealing with SLAs defined on the response-time percentiles. One source of error is that standard methods assume exponentially-distributed processing times, a common assumption that may not capture well the observed distribution of the processing times. Instead, one advantage of our methodology is to consider more general processing times. Figure 1(b) illustrates this disparity by comparing the prediction errors of the exponential assumption and the general processing times we apply in this paper. We further illustrate this and other sources of error in Section VI-B.

A. Modeling Challenges

DBaaS providers have automated fundamental database administrative services, however, this in turn has reduced DBA control on many of the performance aspects of the database [30], especially since service providers do not expose details about collocation, physical resources and data placement. For example, limiting replicas to read-only functionality on Amazon RDS prevents the collection of query-level statistical data from these replicas.[1] Thus the DBA must rely on client-side performance statistics, or on performance measurements from the master database to infer the performance of the replicas. In addition, the DBaaS performance depends on the number of active replicas in the cluster, whether the master DB is currently participating in the creation of a new replica, and the frequency of addition/deletion of replicas from the cluster. Furthermore, reporting the percentiles of performance metrics, in addition to averages, is important for DBAs to accurately determine the impact that a given replication policy has on the QoS perceived by the users.

To cope with these requirements, in this paper we propose a methodology that explicitly models the dynamic nature of replication in DBaaS environments and its impact on the system performance, and provides response-time percentiles for evaluation. To achieve this, we exploit our recent extension [32] of the fluid queueing network model introduced in [14], which incorporates an environment model. The environment model is composed of a number of *stages*, each representing a possible configuration or condition of a system. This concept was tested in [32] to model web-based applications. In this paper, we exploit the environment model for two main purposes: (i) to model how the number of active replicas changes dynamically, increasing and decreasing as part of the operation of the database cluster; and (ii) to capture when the DB master is engaged in a replica creation process, reducing its ability to process incoming queries. We focus on these conditions as they can have a large impact on the cluster performance. In addition, we use Coxian distributions as in [32] to capture the processing times observed at the DB cluster, and we show that this feature is key to obtain accurate estimations of the response-time percentiles, which are commonly used to define SLAs. Section IV describes the model in detail.

[1]At the time of our experiments query-level performance metric collection was unsupported for MySQL read replicas.

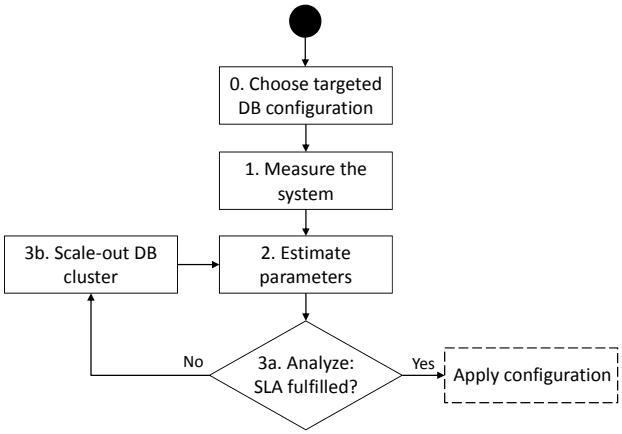


Fig. 2: State diagram of the DBaaS evaluation methodology.

III. Methodology

The general method we propose is to approximate DBaaS service performance via client-side measurements. Here, we use the term *client* to refer to the the source of the workload observed at the database tier. Specifically, we use request response times observed at the client to estimate DBaaS service times. This method simplifies the collection of measurements for DBAs. Our methodology assumes the following characteristics of DBaaS services as currently provided by commercial and public providers [5]–[7]: (i) database server VMs/instances are by default homogeneous; (ii) DBaaS inter-cluster performance metrics are either not accessible or difficult to obtain. In addition, in our case study the client application is hosted on the service provider's IaaS/PaaS infrastructure, which is a common choice that reduces the communication latency between the application tiers.

To model the DB cluster we explicitly consider the changes that the system undergoes in its configuration (replica creation, deletion). We therefore put forward a methodology that collects data for each of these stages and uses it to parameterize a model that computes overall and per stage performance metrics. The steps of our methodology are depicted in Figure 2. Below we give an overview of each step with details provided in the following sections.

1) Measuring the targeted DB cluster: For a given database cluster, run a workload against the master database and initiate a sequence of replica creations/deletions. Collect a client-side trace of request submission times and response times during the execution of the experiment. In addition, an experiment log is kept for all steps in the run, e.g., replica creation start, end, etc. An example experimental case study is detailed in Section V. This step can be completed once for a set of different workloads or by reusing historical data. Sections VI and VII provide additional details.

2) Estimating model parameters: From the request trace we calculate request mean service times and variance for each stage. From the service time mean and variance, we obtain service time distributions. We calculate the mean time the system spends in each stage from the experiment log. This is detailed in Section IV-C.

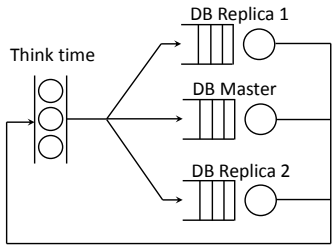


Fig. 3: Queueing network model of the replicated database

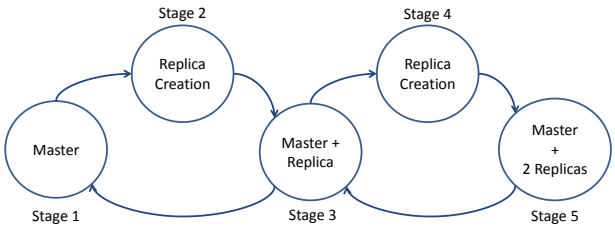


Fig. 4: Example environment model for the replicated database

3) Analyzing the DB cluster: The processing time distributions, mean time spent in each stage, and the number of CPU cores for each DB server are provided as input to the database cluster model introduced in Section IV-A. Solving the model gives the overall and per stage mean response times, throughputs and response time percentiles. The transient overhead caused by the creation of replicas is considered via the environment model discussed in Section IV-B. If the predicted performance fulfills the required SLAs, the DBA can apply the configuration to the cluster. If not, the model is modified to represent the scale out of the DB cluster, especially for those stages where SLA violations are identified. Model parameters may need to be re-estimated for the new configuration, as decribed in Section VII, and the model is again analyzed to verify compliance with the SLAs.

As the measurements needed to parameterize the database replication model are not DBaaS specific the methodology is general enough to apply to any DBaaS framework. We illustrate in Section VIII how our approach can be utilized in automated cloud provisioning tools.

IV. The Database Replication Model

A. The Database Cluster Model

We model a DBaaS environment as a closed queueing network like the one shown in Figure 3. The network is composed of M multi-server stations that represent the database servers. In addition, a delay station is used to model the think time of users submitting requests. Figure 3 illustrates a database cluster with $M = 3$ database servers: one master and two replicas. A total of N users submit requests with an exponentially distributed think time. Requests are distributed equally to all servers in the cluster.

To model fairly general processing times, we rely on Coxian distributions. A Coxian distribution can be interpreted as a sequence of m exponential phases, where all requests start service in phase 1. In phase i a request spends an exponentially-distributed time with rate μ_i, after which it can either complete service, with probability ϕ_i, or continue to the next phase, with probability $1-\phi_i$. After the last phase the request completes service with probability 1. To illustrate the methodology we focus on read requests that belong to a single class, though this can be easily generalized to many classes representing different types of reads and writes. To analyze this model, we follow [32] in using a fluid approximation based on a set of ordinary differential equations, which provides a deterministic approximation to the expected sample path of the system state. Although approximate, the fluid model has the advantage of avoiding the state-space explosion encountered in Markovian models.

B. The Environment Model

We introduce the environment model to capture that (i) the number of active replicas changes dynamically, increasing and decreasing as part of the operation of the database cluster; and that (ii) the DB master can engage in a replica creation process, reducing its ability to process incoming queries. Assuming a maximum of $M-1$ replicas, the environment model consists of $2M-1$ stages, one for each database cluster configuration. An example for the case with two replicas is illustrated in Figure 4. Here, in *stage 1* only the master is running, whereas in *stage 2* a replica is being created while the master continues serving requests. When the replica is ready, the system moves to *stage 3*, where both master and replica are serving requests. If during this stage the replica is deleted, the system goes back to *stage 1*. In a similar manner, for any $1 \leq j \leq M-1$, stage $2j$ refers to a stage with j database servers accepting requests with a replica being created, while in stage $2j+1$ the replica has been created, thus a total of $j+1$ servers are up. Notice that this model can be easily generalized to consider the concurrent creation and deletion of multiple replicas. This can be achieved by adding links and a stage between the relevant stages. For instance, in Figure 4 we can add links between stages 1 and 5, and an intermediate stage, to allow for the creation/deletion of 2 replicas simultaneously.

A key feature of the environment model is to include separate stages for the periods during which replicas are created. The objective of these stages is to alter the query processing times at the master to reflect that the master must devote resources for replica creation, which impacts the request processing times. Also, we modify the routing probabilities whenever a new replica becomes available to allocate incoming requests to the new replica. We set p_i to be the probability that a request is routed to DB server i, such that in stages 1 and 2, where only the master is up, we set $p_1 = 1$ to route all the traffic to the master, letting $p_i = 0$ for $1 < i \leq M$. In stages $2j-1$ and $2j$, where a total of j servers are up, for $2 \leq j \leq M$, we set

$$p_i = \begin{cases} 1/j, & i = 1, \ldots, j, \\ 0, & i = j+1, \ldots, M, \end{cases}$$

such that the traffic is uniformly distributed among the available servers. Although we assume uniform routing, more general routing schemes can be considered.

The database cluster changes its state continually, adapting to the incoming workload, thus adding and deleting replicas at run-time. These dynamics are captured by the environment model by means of a continuous-time Markov chain with as many states as database replication stages. Thus the time spent in stage j is exponentially distributed with mean w_j, and after a visit to stage j, the system jumps to another stage k with

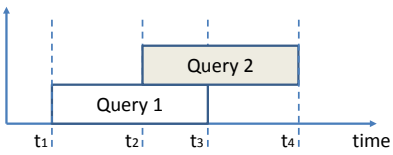


Fig. 5: Example of estimation with the BL method

probability q_{jk}. In the next section, we describe how the model parameters are obtained from measurements.

C. Model parameterization

As stated in Section II, server-side performance monitoring is challenging in DBaaS environments. Our modeling methodology mitigates this scenario by collecting client-side performance statistics, as described in Section V. Since we collect submission timestamps and response times for all clients we can extract service-time samples with the baseline (BL) estimation method introduced in [33], which we illustrate in Figure 5. For the two queries in the figure we collect the submission timestamps t_1 and t_2, and the response times t_3-t_1 and t_4-t_2. If these two queries execute on the same CPU core, they must share the CPU core during the period $[t_2,\ t_3]$. Thus the service time for query 1 is $(t_2-t_1)+(t_3-t_2)/2$ since during the period $[t_2,\ t_3]$ it only uses half of the CPU core processing capacity. In other words, the service time is the time that the query would require to execute if it was processed on its own without sharing CPU resources with other queries. Following a similar analysis for any number of queries and cores, the BL method determines the effective processing time for each query in the trace. Using these service-time samples we obtain their mean and variance, which we then use to estimate a Coxian representation of the service time distribution by means of standard moment-matching methods [34]. These estimates represent the service times of the M database servers in Figure 3.

As the service times can differ among the replication stages, especially due to the replication overhead, we estimate the service time distribution separately for each replication stage, making use of the experiment event log described in Section V. Further, due to the homogeneous configurations of the DB cluster, in which replicas are created with the same resources as the master, we assume replica service demands are equal to those of the master for each stage in which both are serving requests and no replica is being created. For the master server we also estimate its service time when a replica is being created, which is used for this server only during the corresponding stage. To estimate the holding times in each stage in the environment model of Figure 4, we also use the experiment event log, which records the timestamps in which the database cluster adds and removes replicas. We use these timestamps to obtain the duration of all visits to each stage j, which we average to estimate the mean holding time w_j for each stage j. From this log we can also determine the number of times that a transition from each stage to any other stage was observed. We use these counts to estimate the transition probabilities between stages, completing the description of the environment model.

D. The reset matrix

A key component in defining the environment model is the *reset matrix*. When the system transitions from one stage in the environment model to another stage, its state must be modified in order to account for the new configuration. In particular, when the system is in stage $2j-1$, with $j>1$ servers running, and one of the servers is removed, the new system state must be such that all the requests in progress in the server being removed are re-allocated to the other servers. This re-allocation is achieved by means of the reset matrix.

The system state is described by the number of requests executing in each server and in each phase of execution, according to the Coxian service model. We introduce the reset matrix $R_{s,t}$ associated to a transition from stage s to stage t, for $1\leq s,t\leq 2M-1$, such that the entries of $R_{s,t}$ hold the probability that the system jumps from each state in stage s to any state in a different stage t. We can partition the matrix $R_{s,t}$ into blocks $R_{s,t}^{i,k}$ that correspond to transitions from server i to server k. We can then write

$$R_{s,t}=\begin{bmatrix} R_{s,t}^{0,0} & R_{s,t}^{0,1} & \cdots & R_{s,t}^{0,M} \\ R_{s,t}^{1,0} & R_{s,t}^{0,1} & \cdots & R_{s,t}^{1,M} \\ \vdots & \vdots & \ddots & \vdots \\ R_{s,t}^{M,0} & R_{s,t}^{M,1} & \cdots & R_{s,t}^{M,M} \end{bmatrix}.$$

The (j_1,j_2) entry of the $R_{s,t}^{i,k}$ matrix is the probability that a request in execution phase j_1 in server i is re-allocated to execution phase j_2 in server k when the system transitions from stage s to stage t. Now, assume the system is in stage $2j-1$ with $j>1$ servers in operation. When one of these j servers is removed we assume, without lost of generality, that the j-th server is the one being brought down, such that the associated reset matrix is

$$R_{2j-1,2j-3}^{i,k}=\begin{cases} I, & i<j, k=i, \\ \boldsymbol{e}\boldsymbol{e}_1'/(j-1), & i=j, k=1,\ldots,j-1, \\ 0, & \text{otherwise.} \end{cases}$$

The first condition indicates that for any server i different from the server going down (j), the requests simply continue execution in that server without any modification. In contrast, for the server going down (j), its requests are redistributed to the other servers, which correspond to servers 1 to $j-1$. This reallocation is captured by the matrix $\boldsymbol{e}\boldsymbol{e}_1'$, where $\boldsymbol{e}$ is a vector of ones, $\boldsymbol{e}_1$ is a vector with a one in its first entry and zero elsewhere, and $'$ stands for the matrix transpose. Thus, $\boldsymbol{e}\boldsymbol{e}_1'$ is a matrix with a first column of ones and zero elsewhere, so that a request in server j in *any* execution phase re-starts service in another server in the first phase, as all Coxian service times start in the first phase. Further, the requests in the server going down are uniformly distributed across the $j-1$ remaining servers, which is captured by dividing the $\boldsymbol{e}\boldsymbol{e}_1'$ matrix by the factor $(j-1)$. Stage transitions for replica creation can be described similarly, but are simpler since the requests can continue execution in the server in which they are allocated and the new server starts empty.

V. Evaluation Environment Overview

In this section, we describe the software and hardware configurations used in our running case study. We imported

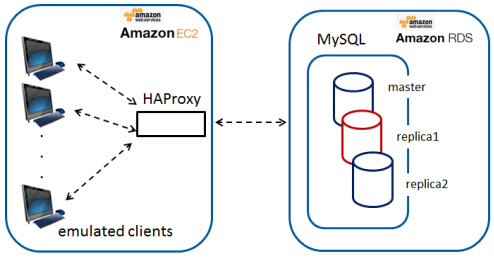


Fig. 6: The Amazon RDS evaluation environment.

TABLE I: Experimental Setup Specifications.

VM	*Specifications*
DB Instance	db.m3.large 2 vCPU 7.5 GiB RAM MySQL version 5.6
Clients VM	m3.large Ubuntu Server 14.04 client mean think time = 0.2s
HAProxy	balance leastconn

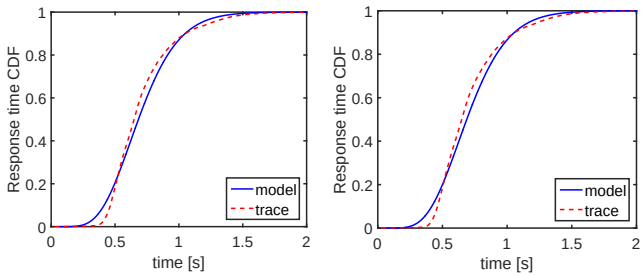
a set of production-level data from an enterprise application[2] onto an Amazon RDS instance hosting MySQL version 5.6, which was the default version provided by Amazon RDS at the time. Amazon RDS provides MySQL replication by managing the creation and configuration of MySQL read replicas. The replicas can only serve read queries, while the write queries must be routed to the master. The MySQL DB used the default configuration provided by Amazon.

As depicted in Figure 6, the workload for the DB cluster is generated from a set of emulated clients, running on VMs deployed on Amazon EC2 [35]. HAProxy [36], an open source load balancer, was installed on the EC2 instance to connect the clients with the MySQL instance(s). The system specification is provided in Table I. The HAProxy load balancer accepts connections from the emulated clients and allocates them, to the master and replicas in the MySQL RDS cluster. During the duration of each experiment, each emulated client loops through the following steps: opens a MySQL connection through HAProxy, sends a read query and waits until receiving the result, closes the connection and waits for an exponentially-distributed think time. We close the connection each time and reconnect to avoid saturating the MySQL connection pool with idle connections. We collect request response times at the EC2 client VM, which include (dis)connection times.

To evaluate the effect of replication on performance, we wrote a script that uses the publicly available Amazon RDS APIs and runs on an EC2 VM to automate the creation/deletion of read replicas during the execution of the experiments and to reconfigure HAProxy each time the DB cluster configuration changed. A log of the experiment was produced to record the time points in which replicas were created/deleted and the availability of the master and replicas during the replication process.

Using the setup described above we conducted experiments with different configurations and number of clients. Since replication is used to handle changes in the workload, in the experiments we consider two scenarios: (i) unscaled, where the workload (number of users) remains unchanged during the duration of the experiment; (ii) scaled, where the workload is increased/decreased proportionally to the number of active replicas. In addition, we consider two modes of evolution of the cluster configuration: (i) deterministic, where the configuration changes are predefined, i.e., the times of addition/deletion of replicas are defined at the beginning of the experiment; (ii) probabilistic, where the configuration changes randomly, adding/deleting replicas so as to reflect online adaptations. The details of the experimental setups are in Table II.

[2]Due to a non-disclosure agreement we are unable to provide details of the application, but it has a standard three-tier architecture with presentation, application and data tiers.



(a) Stage 1 (b) Stage 2

(c) Stage 3 (d) Overall

Fig. 7: $N = 90$ - Response time CDF for each stage for the `uM1R` experiment vs. empirical CDF.

VI. Validation

In this section we evaluate the ability of the proposed model to predict response times similar to those observed in the experiments described in Section V. We also investigate the model sensitivity to the assumptions described in Section IV.

A. Master and One Replica

1) Fixed Workload: To validate the replication model we compare the results of the model against the experimental results of the `uM1R` setup. This scenario has the first three stages in Figure 4 and we parameterize the model from the measurements of the `uM1R` setup with 90 clients. From experimentation, this workload maintains the mean server utilization at 80% when only the master DB is running. Figure 7 shows the cumulative distribution function (CDF) of the response time observed (trace) and the one predicted by the model. Figure 7(a) shows the response time CDF for the first stage of the system, where only the master is running; Figure 7(b) for stage 2 during the replication phase; and Figure 7(c) for stage 3 after replica creation; and Figure 7(d) illustrates the overall response-time CDF.

For all cases, we observe that the predicted response time CDFs approximate well the observed response time CDFs for

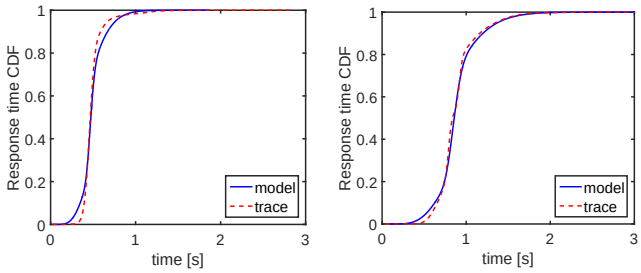
TABLE II: Description of Experiments.

Experiment	*Description*	*Duration*
unscaled Master and 1 Replica (`uM1R`)	(1) fixed number of clients run against the master DB for 5 min (2) create a read replica (3) after the replica is ready to accept connections, continue for another 10 min.	30 min
scaled Master and 1 Replica (`sM1R`)	(1) conduct the `uM1R` experiment (2) when the replica is ready, scale the workload proportional to the server capacity	30 min
unscaled Master and 2 Replicas (`uM2R`)	(1) run a fixed number of clients against the master DB for 5 min (2) create a read replica (3) after the replica is ready to accept connections, continue for 5 min (4) create the second replica from the master	50 min
scaled Master and 2 Replicas (`sM2R`)	(1) conduct the `uM2R` experiment (2) each time a replica is ready, scale the workload proportional to the available DB servers	50 min
unscaled probabilistic Master and 2 Replicas (`upM2R`)	(1) run a fixed number of clients and create and delete up to 2 replicas multiple times	3 hours
scaled probabilistic Master and 2 Replicas (`spM2R`)	(1) conduct the `upM2R` experiment (2) scale the workload in and out proportionally to the number of available DB servers	3 hours

all stages and for the overall response times. While the errors for the *mean* response time are at most 1%, Table III (column 90) shows that the mean relative error in the CDF, measured at every percentile, is at most 6%, and Tables IV and V (column 90) show the relative error for the 95^{th} and 99^{th} percentiles, which are at most 8%. We observe that the model is able to predict not only the average metrics but also the dispersion of the response times, which change depending on the number of DB servers available.

As our model distinctly incorporates the performance of each stage of the system, we are able to observe key changes in the response-time distributions across the different stages, observations that are backed by the measurements. In particular, the response times have a minimum value, or a lower bound, in each stage, but the value of this bound depends on the stage, and particularly on the number of active replicas. While Figures 7(a) and (b) display a similar lower bound, this bound increases in Figure 7 (c), where the DB replica becomes available. When evaluating the overall response times, as depicted in Figure 7 (d), this effect is masked, since the overall response time CDF combines all three stages. This effect is also visible by comparing the sigmoid shape of the CDF in stage 1 against that in stage 3, which is closer to a step function. The proposed model is therefore able to reflect these detailed changes in the response-time distribution, which are also visible in the measurements, as the system configuration changes.

2) Different Workloads: We now consider a more challenging evaluation by using the processing times estimated when the number of clients is N=90 to determine the response times when the number of clients N varies between 70 and 170. The overall response time CDF for $N = 70, 110, 130, 150$ is depicted in Figure 8, where we observe an excellent match between the empirical distribution and the model results. The model captures how the increase in the number of clients shifts the response times to the right and at the same time flattens the distribution. At N=70 the response times are highly concentrated around 0.5 s, while at N=150 most response times are between 1 and 2 s, with more pronounced left and right tails. Figure 9 illustrates the response time CDF for the specific case N=130 for each stage and overall, comparing the predicted CDF obtained with the N=90 measurements against the empirical CDF. We observe the excellent match between the predicted and measured CDFs, with the model clearly capturing the effect of the system configuration as the



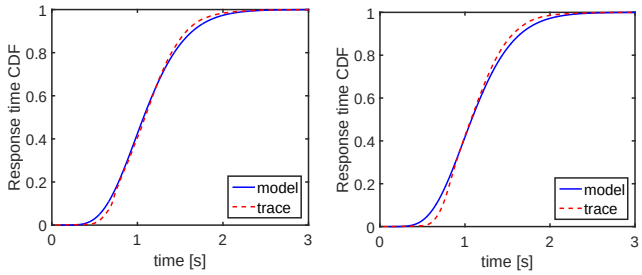
(a) $N = 70$ (b) $N = 110$

(c) $N = 130$ (d) $N = 150$

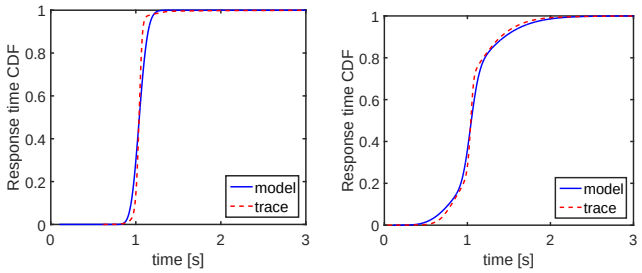
Fig. 8: Response time CDF for $N = 70, 110, 130, \& 150$ based on $N = 90$ observations

number of replicas increases, which clearly affects not just the mean but the distribution as a whole. The ability of the model to capture these changes in the distribution is key to determine possible violations in SLAs specified as percentiles.

In more detail, the predictions errors for the mean response time for each stage and overall are at most 1% for all workloads. Table III shows the mean predictions errors for the response time percentiles. The model gives excellent prediction for all workloads and stages with relatively less accuracy for $N = 170$ for stage 3 with an error of 16%. Tables IV and Table V show the predictions errors for the 95^{th} and 99^{th} percentile, respectively. We observe similar accuracy for these percentiles as for the overall CDF. However, for N=70 we observe a decrease in accuracy for the 99^{th} percentile prediction. For N=170, the model underestimates the 95^{th} and 99^{th} percentiles, especially for stage 3. This can be a result of the increased variance in response times when N=170 in comparison to that when $N = 90$ which is not captured by the model. We will examine further the effect of the processing time variance when considering scaled workloads in Section VII. In contrast, the overall 95^{th} and



(a) Stage 1 (b) Stage 2



(c) Stage 3 (d) Overall

Fig. 9: Response time CDF for $N = 130$ based on $N = 90$ observations

TABLE III: Mean Prediction Errors in Response Time CDF Percentiles(%) based on $N = 90$ for $N = 70 - 170$

N	*70*	*90*	*110*	*130*	*150*	*170*
Stage 1	10	6	3	3	6	7
Stage 2	12	6	5	5	6	8
Stage 3	0	1	2	3	3	16
Overall	6	3	3	3	3	4

99^{th} response time percentiles prediction is accurate for all workloads. In general, the relative errors in the response-time percentiles are small, even though the processing-time distribution is approximated by means of two moments only.

B. Sensitivity Analysis

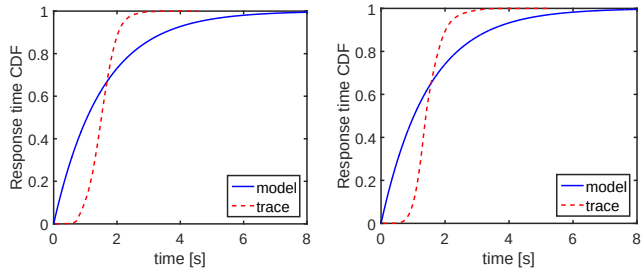
We now evaluate the sensitivity of our modeling approach to the modeling assumptions of our methodology. In particular, we evaluate the accuracy of the model under simpler processing time distributions and under processing times that average out the effect of the replication stages. In both cases, we utilize the same environment model as in the previous section, as removing the environment model would produce

TABLE IV: Prediction Errors in 95^{th} Response Time Percentile(%) based on $N = 90$ for $N = 70 - 170$

N	*70*	*90*	*110*	*130*	*150*	*170*
Stage 1	0	6	0	6	12	13
Stage 2	6	6	2	7	13	16
Stage 3	0	1	2	7	7	33
Overall	13	1	2	5	7	1

TABLE V: Prediction Errors in 99^{th} Response Time Percentile(%) based on $N = 90$ for $N = 70 - 170$

N	*70*	*90*	*110*	*130*	*150*	*170*
Stage 1	18	8	0	5	12	14
Stage 2	15	8	3	8	16	19
Stage 3	1	1	4	6	30	46
Overall	16	7	2	7	9	2



(a) Stage 1 (b) Overall

Fig. 10: Exponential service times - $N = 90$ - Response time CDF for the `uM1R` experiment.

a simple closed queueing model that can only represent the overall performance of the system.

a) Exponential service times: We first consider a model where the processing times follow the standard exponential distribution rather than the Coxian distribution described in Section IV. The exponential distribution is parameterized by the *mean* processing time obtained with the method described in Section IV, but ignoring higher moments. As before, we use the data for the `uM1R` experiment with N=90 clients to obtain the response time distribution for the same and other client numbers. The exponential model produces results that accurately match the *mean* response time, with errors of up to 1%. However, the errors in the response time percentiles are very large, e.g., the mean error in the response time percentiles is 49% for stage 1, rising up to 50% and 62% for stages 2 and 3, respectively. The mean error overall is 52%. For the 95^{th} percentile, the case is more extreme, with the error being 89%, 94% and 151% for stages 1, 2 and 3, respectively. The overall error for the 95^{th} percentile is 105%.

Figure 10 depicts the response-time CDF observed against the one obtained with the exponential model. Clearly, the response time distribution predicted by the model with exponential processing times is very far from the measurements, specifically the tail is much longer, giving rise to very large errors for the high percentiles. This holds for each stage separately and for the overall metrics. These results highlight the importance of considering Coxian processing times to obtain accurate estimations of the response-time percentiles. In fact, while the *mean* response time in a processor-sharing queue is known to be insensitive to the processing time distribution, the response time *distribution* is not, yet this distribution is central for SLA evaluation. This also rules out the use of standard methods that focus on the mean response time only.

b) Average service times: We now consider an alternative model where the service times are not characterized for each stage. Instead, we obtain a single service-time distribution from the *overall* statistics (mean and variance), which is used in every stage. Notice that we keep the environment model in order to modify the number of active replicas in each stage. As a result, the only difference across stages is in the number of active DB servers. Our objective is to test the relevance of considering *different* service-time distributions for each stage as done in the proposed methodology. This model results in very large errors for the mean response time, in the $30 - 60\%$ range, when the stages are considered separately. The errors are much smaller when considering the overall system, on average

15%, as the overall modeling of the service times is better suited to address the overall system performance rather than each stage separately. However, this error is much larger than the 1% error of the proposed model. Further, the errors for the 99th response-time percentile obtained with the modified model average between $56\% - 89\%$ for all stages and 78% overall, which are clearly much larger that the errors produced by the proposed model.

We have shown that the power of our modeling approach depends on the environment model that differentiates the service times observed in each stage, and hence simply averaging the service times produces incorrect results. In addition, capturing the service time distribution, via a Coxian representation, rather than just the mean as with with a standard exponential distribution, is essential in obtaining accurate results for the response-time percentiles.

VII. Scaled and Dynamic Replication

We now consider the scenarios with scaled workloads and dynamic replication, as described in Table II.

A. Scaled sM1R

We consider the sM1R experiment with N=90 clients, in which the number of clients is doubled when the replica is ready to accept requests. When parameterizing the model from the measurements of the sM1R experiment the model gave excellent accuracy for the response time mean and percentiles. As this experiment differs from the uM1R experiment only by the doubling of the workload during stage 3, we modified the model by parameterizing it with the uM1R data, and doubled the workload when the system entered stage 3.

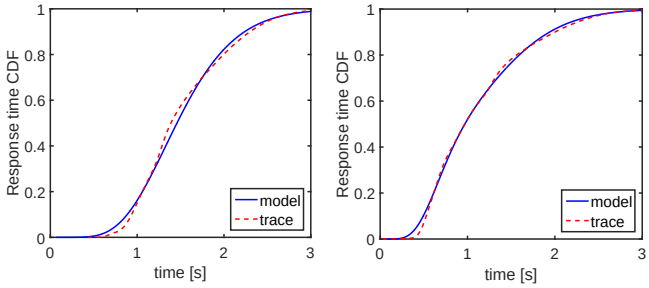
Due to the scaling of the workload, the variability observed in stage 3 of sM1R is much larger than the one observed in the same stage of uM1R, where the number of users is not scaled in the last stage. We note that the workload in stage 3 of sM1R (a total of 180 users) divided equally between the master and the replica is similar to the workload on the master alone in stage 1 (90 users) of the uM1R. To account for this in the model, we use the second moment observed in the first stage of the uM1R setup to parameterize the service times in all stages when modeling the sM1R setup. Thus, although we change the mean demand in each stage as observed in the uM1R setup, we keep the same variance as the one observed in the first stage of uM1R.

From Table VI, we observe that the errors for stage 3 are similar to those for the other stages in both the mean and the percentiles, with similar accuracy to the results presented in the previous section. Figure 11 compares the response time CDF predicted by the modified model against the traces of the sM1R experiment. We observe that the accuracy of the performance prediction depends on the accurate representation of the variability in the service demands as the second moment appears to depend heavily on the ratio of the number of users to the number of resources.

For comparison we show in Figure 12 the response time CDF prediction obtained when we use the variance observed in each phase in uM1R for the corresponding phase in sM1R. Figure 12(a) shows the prominent difference for stage 3, as the predicted response times are much more concentrated than the measurements. Figure 12(b) shows how this affects the overall predictions as well. Therefore, to accurately parameterize the model the second moment plays an important role and should be set according to the ratio between the number of users and the number of active servers in a given stage. To this end, it is sufficient to measure the variability for a range of diverse workloads on a single database instance, then, for predicting a certain scenario we compute the ratio of expected number of users to active DB servers, and obtain the appropriate variance by interpolating the closest values among the observations collected.

TABLE VI: Errors (%) in response time prediction for sM1R based on uM1R

	mean RT	*RT CDF*	*RT95*	*RT99*
Stage 1	< 1	4.2	3.1	5.7
Stage 2	1	7.2	< 1	2.2
Stage 3	< 1	3.2	1	3.3
Overall	< 1	2.9	3	1.1



(a) Stage 3 (b) Overall

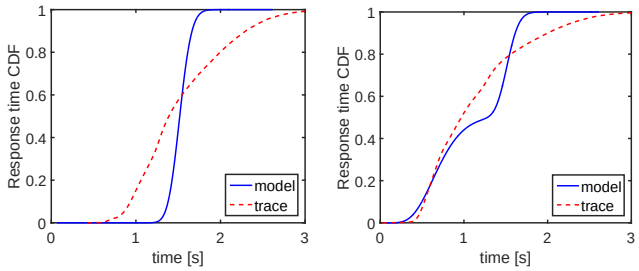
Fig. 11: Response time CDF for sM1R $N = 90$ based on uM1R $N = 90$ observations

B. Dynamic Replication: Master and up to Two Replicas

When experimenting with clusters with 2 replicas we observed the expected decrease in mean response times as the number of replicas increased. However, due to MySQL synchronization overhead between the master and replicas [37], the decrease seemed small and far from proportional to the number of replicas. As we perform client-side measurements that avoid directly measuring the replication overhead, to predict the performance of clusters of size n we require client-side measurements from a cluster of the same size.

In the following, we demonstrate the ability of the model to predict the performance metrics under dynamic replication, i.e., the cluster dynamically adds and removes replicas, from measurements from a static experiment. We show results for up to 2 replicas, but the methodology can be easily applied to any number of replicas, including the 5 replica limit on Amazon RDS.

1) Unscaled upM2R: Here we parameterize the model with data from the uM2R experiment, where replicas are spawned deterministically, to determine the response times in the conditions of the upM2R experiment, where up to 2 replicas are added and removed dynamically. We also use the mean times the cluster was observed in each stage in the upM2R setup, as this information is specific for each particular scenario and cannot be extrapolated. With these parameters we obtain the



(a) Stage 3 (b) Overall

Fig. 12: Response time CDF for sM1R $N = 90$ based on uM1R $N = 90$ observations - without variance correction

TABLE VII: Errors (%) in response time prediction for upM2R based on uM2R

	mean RT	*RT CDF*	*RT95*	*RT99*
Stage 1	2.6	5.4	2.5	1.7
Stage 2	< 1	4.4	4.7	8.8
Stage 3	1.2	1.5	3.2	5.3
Stage 4	< 1	1.8	2.4	2.4
Stage 5	< 1	2.4	6.5	9.5
Overall	< 1	5.1	3.5	8.4

response time CDF per stage and overall and compare them against the observations from the upM2R case with N=90 clients. The model consistently matches the observed response time distribution for all the system stages. Also, the use of the mean times in each stage allows the model to obtain an accurate picture of the overall response times. The relative errors in response-time mean and percentiles are summarized in Table VII, where we observe errors always below 10% even for the highest percentiles.

2) Scaled spM2R: We now conduct a similar experiment for the scaled workload setup. We use the data from the uM2R experiment, with 90 clients, to estimate the processing time distribution, and use these parameters to execute the model in the conditions of the spM2R experiment. Together with the demands, we use the mean times spent in each stage as observed in the dynamic setup spM2R to parameterize the environment model. As the uM2R setup has an unscaled workload, we use the service time variance observed in the first stage of uM2R to parameterize the service times for all stages of spM2R, preserving the resources-to-users ratio, as all stages of spM2R have 90 users per resource. As depicted in Table VIII, we observe an very good match between the trace and the model, both for each stage and overall. Also, we observe a larger error, 9.7%, in the mean response time, although the errors for the percentiles are similar to the cases considered before. We note that the error for stage 5 for the 99-th percentile is 20.7%, which can be attributed to the high variance in stage 5.

TABLE VIII: Errors (%) in response time prediction for spM2R based on uM2R

	mean RT	*RT CDF*	*RT95*	*RT99*
Stage 1	3.3	4.6	3.9	5.9
Stage 2	< 1	4.9	4.7	6.4
Stage 3	2.4	4.6	1.8	2.0
Stage 4	< 1	7.8	5.3	12.1
Stage 5	< 1	6.3	13.5	20.7
Overall	9.7	14.2	8.4	16.4

VIII. Replication Trade-offs

In this section, we evaluate the impact of replication overhead and speed on the response times and demonstrate the applicability of our methodology in automating scaling decisions for DBaaS replication.

A. The Effect of Replication Overhead

As illustrated in Section II, replication can significantly increase the transaction processing time due to the overhead posed on the master DB server during the replica creation phase. Here we make use of the proposed model to determine the impact that different levels of overhead can have on the response times offered by the database. We parameterize the model with the uM1R experimental dataset with N=70 clients, but modify the *mean* transaction service times during stage 2, which is the stage when the replica is being spawned, and the master is the only server effectively processing requests. The mean transaction service time during stage 2 is set as x% more than the mean transaction service time during stage 1. We consider cases up to $x = 20$, in agreement with the observations in Section II. We note that the system spends only 5% of the total execution time in the replication phase.

Figure 13 illustrates the increase in the response times, during the replication phase and overall, as a function of the overhead in the replication stage, comparing against the baseline without overhead. Figure 13(a) shows that the overhead affects both the mean and high percentiles, and that, when considering the replication stage on its own, the increase is larger than the overhead introduced. For example, when the overhead is 15%, the mean response time during the replication phase increases by 21%, while the high response time percentiles increase by around 19%.

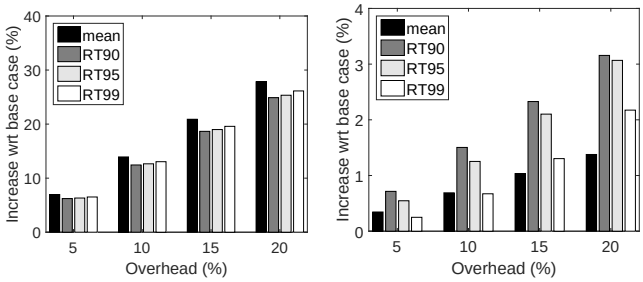
Figure 13(b) considers the effect on the *overall* response times. The 90$^{\text{th}}$ and 95$^{\text{th}}$ percentiles present the largest increase, which is over 3% when the overhead is 20%, while the 99$^{\text{th}}$ percentile increases by 2.3% and the mean by 1.4%. In this case the impact of the replication overhead on the response time percentiles is more apparent than on the mean, which reflects that the requests with the longest response times, overall, are those processed during the replication phase, which are therefore more susceptible to the replication overhead. Also, an increase of 3% in the 90$^{\text{th}}$ percentile is actually quite significant when we consider that stage 2 represents less than 5% of a cycle made of 3 stages. Validation of these results against trace-drive simulations, shown in Table IX, show errors for the mean of around 1% and for the high percentiles below 16%. Therefore, the replication overhead has a large impact on the response times during the replication phase, which can lead to SLA violations during this period. Moreover, this overhead can impact the *overall* response times, particularly the tail percentiles, harming the requests that experience the longest response times.

B. The Effect of Replication Speed

Depending on a number factors, e.g., network speed, DB master image size, or log file size, the replica creation phase can take a substantial amount of time. While replicas can come up and down, we are interested in the proportion of time that the system is under replication, as this is the period when the

TABLE IX: Error in overall response times: model prediction vs. trace-driven simulation

Overhead (%)	*mean RT (%)*	*RT90 (%)*	*RT95 (%)*	*RT99 (%)*
0	1.2	15.8	12.2	11.1
5	1.2	15.4	11.9	11.3
10	1.1	14.7	11.3	11.2
15	1.0	14.1	10.6	10.7
20	0.9	13.5	9.9	10.2



(a) During replication (b) Overall

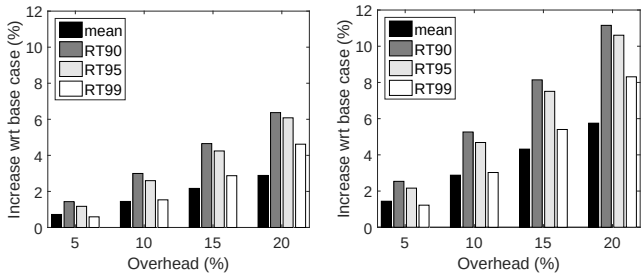
Fig. 13: Increase in response times as overhead increases

DB master server suffers additional pressure. In the previous section, the replication phase was approximately 5% of the total execution time. We increase the state holding time for stage 2 such that the total time that the database is in the replication phase is 10% and 20% of the overall execution time. We evaluate the effect of the replication time as we vary the replication overhead up to 20%, as in the previous section. We focus on the overall metrics since the metrics *during* the replication stage are not affected by the duration of this stage.

Figures 14(a)-(b) depict the percentage increase in overall mean response time mean, 90th, 95th and 99th percentiles, when the system replicates 10% and 20% of time, respectively, compared to the base case without overhead. As in the previous section, we observe that the effect of the overhead during replication is higher for the high percentiles than for the mean. However, the impact on the mean clearly increases as the replication time increases. For instance, in the previous section the increase in the mean response time under a 20% overhead was just 1.4%, while here it is 2.8% and 5.7% when the replication times are 10% and 20%, respectively. Validation of these scenarios against trace-driven simulations show errors similar to those in Table IX, we have omitted the numbers due to space limitations. We also observe that the impact on the high percentiles is more pronounced as the replication time increases, and, of the three percentiles considered, the highest impact is on the 90th. Thus, while under a small replication time the impact of replication is felt only by the users that face the highest response times, for large replication times this impact becomes more widespread, affecting more significantly the average metrics. The effect of replication in this case thus moves from the tail down to the main body of the response-time distribution, although the highest percentiles are still the ones most affected by the replication overhead.

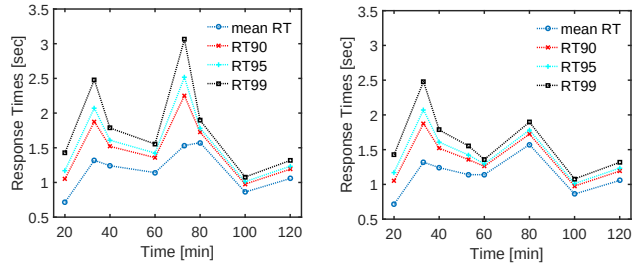
C. Guiding Scaling Decisions

Our methodology can be easily plugged into current automated cloud provisioning tools, as the inputs to the models are easily available to these tools. To illustrate this, consider a DBaaS cluster where provisioning decisions are made every



(a) Replication time: 10% (b) Replication time: 20%

Fig. 14: Effect of replication overhead and time



(a) Scaling scenario 1 (b) Scaling scenario 2

Fig. 15: Evaluating scaling decisions based on response time predictions

20 minutes. To this end a forecast of the expected workload, in intervals of 20 minutes, is maintained, e.g., from historical data. The provisioning tool then evaluates possible scenarios on whether and when to create or delete replicas.

Let us consider an initial state of the cluster with the master server only, and a forecast of $\{90, 150, 140, 180, 110, 130\}$ expected users in the next six 20 minute intervals of the forecasting horizon. The provisioning tool will compare between different scaling scenarios to achieve the best performance for the forecast workload. As an example, consider 2 possible scaling scenarios: scenario (1) $\{1, 2, 2, 3, 2, 2\}$, in which the cluster is scaled out to 2 replicas in the second interval, then scaled out to 3 replicas in the fourth interval, and then scaled in to 2 servers in the fifth interval; and scenario (2) $\{1, 2, 3, 3, 2, 2\}$, where the only difference with scenario 1 is in whether a 3rd server should be started at the beginning of the third instead of the fourth period. The expected response times, mean and high percentiles, are depicted in Figure 15 for these two scenarios. Here the performance metrics for a given interval are shown at the end of the interval, e.g, the statistics of the response times for the first interval are marked at minute 20. Also, note that the creation of a replica divides a planning interval into two phases: during replication, and after replication. For example, in Figure 15 a second server is started in the second interval, thus the response times just after minute 20 increase due to replica creation and the increased workload, but decrease at around minute 33, when the second server is ready. Comparing the two scenarios in Figure 15, scenario (2) offers shorter response times, especially for the percentiles, due to the launching of the 3rd server *before* the beginning of interval 4, which has the peak number of users, 180. Cloud provisioning tools can perform these comparisons automatically to obtain the *best* scaling scenario, where the

choice differs depending on the specific objective, e.g., finding the scenario with the minimum cost that attains the predefined SLAs in the response-time percentiles during the operation of the system. The methodology proposed is well-suited to support these scaling decisions, as it can efficiently evaluate the response-time percentiles for many scenarios explicitly considering the impact of replication.

IX. Conclusion

In this paper we have introduced a measurement-driven methodology for evaluating the impact of replication on the QoS of relational DBaaS offerings. The methodology builds upon an analytical model representing the database cluster configurations combined with an environment model to represent the transient replication stages. In comparison to experiments on Amazon RDS and trace-driven simulation we have shown the accuracy in predicting not only the *mean* response time, but also the *percentiles* of the response-time distribution, which are central for evaluating SLAs.

We plan to investigate more complex modeling scenarios, including mixed workloads and multiple replica creation and deletion. In addition, we will investigate, through experimentation, the relationship between workload variance, synchronization overhead and workload characteristics. We anticipate that replication for performance will be further adopted by DBaaS users and thus anticipate the need for such a methodology within current cloud provisioning tools.

Acknowledgment

The research of Rasha Osman and Juan F. Pérez leading to these results has received funding from the EPSRC grant EP/M009211/1 (OptiMAM). The research of Giuliano Casale was supported by the European Union under grant agreement H2020-644869 (DICE). The research of Juan F. Pérez is supported by the ARC Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS).

References

[1] G. Linden, "Make data useful." [Online]. Available: http://www.gduchamp.com/media/StanfordDataMining.2006-11-28.pdf

[2] G. Linden and M. Mayer, "Presented at web 2.0. 2006." [Online]. Available: http://bit.ly/1OkDCM8

[3] E. Schurman and J. Brutlag., "Performance related changes and their user impact," Velocity Web Performance and Operations Conference, 2006.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 205–220, 2007.

[5] Amazon Web Services, "RDS." [Online]. Available: https://aws.amazon.com/rds/

[6] Microsoft. Microsoft Azure, "https://azure.microsoft.com/en-gb/."

[7] Google Cloud SQL, "https://cloud.google.com/sql/."

[8] Oracle, "Oracle database cloud." [Online]. Available: http://www.oracle.com/technetwork/database/database-cloud/index.html

[9] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Perform. Eval.*, vol. 67, pp. 634 – 658, 2010.

[10] R. Osman and W. J. Knottenbelt, "Database system performance evaluation models: A survey," *Perform. Eval.*, vol. 69, pp. 471–493, 2012.

[11] Object Management Group (OMG), "A UML profile for MARTE: Modeling and analysis of real-time embedded systems, beta 2," OMG, Tech. Rep. OMG Document Number: ptc/2008-06-09, 2008.

[12] M. Tribastone, P. Mayer, and M. Wirsing, "Performance prediction of service-oriented systems with layered queueing networks," in *ISoLA*, 2010.

[13] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, pp. 3 – 22, 2009.

[14] J. F. Pérez and G. Casale, "Assessing SLA compliance from palladio component models," in *SYNASC*, 2013.

[15] D. Dubois and G. Casale, "Autonomic provisioning and application mapping on spot cloud resources," in *IEEE ICCAC*, 2015.

[16] Amazon Web Services, "Working with PostgreSQL, MySQL, and MariaDB read replicas." [Online]. Available: http://amzn.to/1MVKwD7

[17] M. Klems, D. Bermbach, and R. Weinert, "A runtime quality measurement framework for cloud database service systems," in *QUATIC*, Sept 2012, pp. 38–46.

[18] T. Dory, B. Mejias, P. V. Roy, and N. L. Tran, "Measuring elasticity for cloud databases," in *Cloud Computing*, 2011.

[19] J. Kuhlenkamp, M. Klems, and O. Röss, "Benchmarking scalability and elasticity of distributed database systems," *VLDB Endow.*, vol. 7, no. 12, pp. 1219–1230, Aug. 2014.

[20] P. Di Sanzo, R. Palmieri, B. Ciciani, F. Quaglia, and P. Romano, "Analytical modeling of lock-based concurrency control with arbitrary transaction data access patterns," in *WOSP/SIPEW*, 2010.

[21] R. Osman, D. Coulden, and W. J. Knottenbelt, "Performance modelling of concurrency control schemes for relational databases," in *ASMTA*, 2013.

[22] P. Merkle and C. Stier, "Modelling database lock-contention in architecture-level performance simulation," in *ACM/SPEC ICPE*, 2014.

[23] M. Nicola and M. Jarke, "Performance modeling of distributed and replicated databases," *IEEE Trans. Knowl. Data Eng.*, vol. 12, pp. 645–672, 2000.

[24] S. Elnikety, S. Dropsho, E. Cecchet, and W. Zwaenepoel, "Predicting replicated database scalability from standalone database profiling," in *EuroSys*, 2009.

[25] D. Menasce, H. Gomaa, S. Malek, and J. Sousa, "Sassy: A framework for self-architecting service-oriented systems," *Software, IEEE*, vol. 28, pp. 78–85, 2011.

[26] A. Koziolek, H. Koziolek, and R. Reussner, "Peropteryx: Automated application of tactics in multi-objective software architecture optimization," in *QoSA-ISARCS '11*, 2011, pp. 33–42.

[27] A. Antoniou, "Performance evaluation of cloud infrastructure using complex workloads," Master's thesis, Delft University of Technology, 2012.

[28] D. Franceschelli, D. Ardagna, M. Ciavotta, and E. Di Nitto, "Space4cloud: A tool for system performance and costevaluation of cloud systems," in *MultiCloud '13*, 2013, pp. 27–34.

[29] G. Casale, M. Tribastone, and P. G. Harrison, "Blending randomness in closed queueing network models," *Perform. Eval*, vol. 82, pp. 15 – 38, 2014.

[30] T. Kiefer, H. Schön, D. Habich, and W. Lehner, "A query, a minute: Evaluating performance isolation in cloud databases," in *Performance Characterization and Benchmarking. Traditional to Big Data*, 2015.

[31] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[32] J. F. Pérez and G. Casale, "Line: A scalable tool for evaluating software applications in unreliable environments," under review.

[33] J. F. Pérez, G. Casale, and S. Pacheco-Sanchez, "Estimating computational requirements in multi-threaded applications," *IEEE Trans. Softw. Eng.*, vol. 41, pp. 264 – 278, 2015.

[34] W. Whitt, "Approximating a point process by a renewal process, I: Two basic methods," *Oper. Res.*, vol. 30, pp. 125–147, 1982.

[35] Amazon Web Services, "EC2." [Online]. Available: https://aws.amazon.com/ec2/

[36] HAProxy, "http://www.haproxy.org/." [Online]. Available: http://www.haproxy.org/

[37] MySQL, "Improving replication performance." [Online]. Available: http://bit.ly/1HGxa0R