Application-Managed Database Replication on Virtualized Cloud Environments

Liang Zhao \(^*\), Sherif Sakr \(^*\†\), Alan Fekete ‡, Hiroshi Wada \(^*\†\), Anna Liu \(^*\†\)

\(^*\)National ICT Australia (NICTA)
\(^†\)School of Computer Science and Engineering
University of New South Wales, Australia
{firstname.lastname}@nicta.com.au

\‡School of Information Technologies
University of Sydney, Australia
{firstname.lastname}@sydney.edu.au

Abstract—One among several patterns that are common for applications being deployed in cloud platforms is to take an existing application designed for a conventional data center, and then port it to the cloud with minimal changes. When this is done, the application tier can easily take advantage of the elasticity and scale provided by the cloud, but the data management layer, being stateful, faces more issues. In this paper, we explore experimentally the limits to scaling for an application that itself manages database replicas each placed in a virtual machine in the cloud (exactly following the design used when the application would be deployed on an in-house cluster). We characterize important limits in the load on the master copy, the workload imposed on each slave copy when processing updates from the master, and also from the increasing staleness of replicas.

I. INTRODUCTION

The cloud is an increasingly popular platform for hosting software applications in a variety of domains such as e-retail, finance, news and social networking. Cloud computing simplifies the time-consuming processes of hardware provisioning, hardware purchasing and software deployment. Therefore, it promises a number of advantages for the deployment of data-intensive applications such as elasticity of resources, pay-per-use cost model, low time to market and the perception of (virtually) unlimited resources and infinite scalability. Hence, it becomes possible, at least theoretically, to achieve unlimited throughput by continuously adding computing resources (e.g. database servers) if the workload increases.

In general, virtual machine technologies are increasingly being used to improve the manageability of software systems and lower their total cost of ownership. Resource virtualization technologies add a flexible and programmable layer of software between applications and the resources used by these applications. One approach for deploying data-intensive applications in cloud platforms is the application-managed approach which takes an existing application designed for a conventional data center, and then port it to virtual machines in the public cloud. Such migration process usually requires minimal changes in the architecture or the code of the deployed application [1]. In this approach, database servers, like any other software components, are migrated to run in virtual machines. One of the main advantages of the application-managed approach is that the application can have the full control in dynamically allocating and configuring the physical resources of the database tier (database servers) as needed [2], [3], [4], [5], [6]. Hence, software applications can fully utilize the elasticity feature of the cloud environment to achieve their defined and customized scalability or cost reduction goals. In addition, this approach enables the software applications to build their geographically distributed database clusters. Without the cloud, building such in-house cluster would require self-owned infrastructure which represent an option that can be only afforded by big enterprises.

The application-managed approach is not the only way to have scalable persistent data in the cloud. Two other approaches are widespread. The platform-supplied storage layer approach relies on a new wave of storage platforms named as key-value stores or NoSQL (Not Only SQL) systems. These systems are designed to achieve high throughput and high availability by giving up some functionalities that traditional database systems offer such as joins and ACID transactions [7]. For example, most of NoSQL systems offer weaker consistency properties (e.g. eventual consistency [8]). Cloud offerings of this scenario include Amazon S3\(^1\), Amazon SimpleDB\(^2\) and Microsoft Azure Table Storage\(^3\). In practice, migrating existing software application that uses relational database to NoSQL offerings would require substantial changes in the software code due to the differences in the data model and query interface. In addition, developing applications on top of an eventually consistent datastore requires a higher effort compared to traditional databases [9].

The relational database as a service is another approach in which a third party service provider hosts a relational database as a service [10]. Such services alleviate the need for their users to purchase expensive hardware and software, deal with software upgrades and hire professionals for administrative

\(^1\)http://aws.amazon.com/s3/
\(^2\)http://aws.amazon.com/simpledb/
and maintenance tasks. Cloud offerings of this scenario include Amazon RDS\(^4\) and Microsoft SQL Azure\(^5\). For example, Amazon RDS provides access to the capabilities of MySQL or Oracle database while Microsoft SQL Azure has been built on Microsoft SQL Server technologies. As such, users of these services can leverage the capabilities of traditional relational database systems such as creating, accessing and manipulating tables, views, indexes, roles, stored procedures, triggers and functions. It can also execute complex queries and joins across multiple tables. The migration of the database tier of any software application to a relational database service is supposed to require minimal effort if the underlying RDBMS of the existing software application is compatible with the offered service. However, many relational database systems are not, yet, supported by the DaaS paradigm (e.g. DB2, Postgres). In addition, some limitations or restrictions might be introduced by the service provider for different reasons\(^6\).

A common feature to the different cloud offerings of the platform-supplied storage services and the relational database services is the creation and management of multiple replicas of the stored data while a replication architecture is running behind-the-scenes to enable automatic failover management and ensure high availability of the service. In our previous work [9], we have experimentally investigated consumer-based observations of the consistency, data staleness and performance properties of various cloud NoSQL offerings. In this paper, we focus on the application-managed approach. Our aim is to set a first yardstone in evaluating the performance characteristics of database replication in virtualized cloud environment. Using the database and different workloads of the Cloudstone benchmark [11], we experimentally assess the behavior of the master-slave database replication strategy on Amazon EC2. In particular, we focus on addressing the following questions:

- How well do the master-slave replication strategy scale with an increasing workload and an increasing number of database replicas in a virtualized environment? In principle, we try to understand what factors act as limits on achievable scale.
- What is the average replication delay (window of data staleness) that could exist with an increasing number of database replicas and different configurations to the geographical locations of the slave databases?

To the best of our knowledge, this is the first work providing a performance study of deploying application-managed replicated database tier in public virtualized cloud environment. The remainder of this paper is structured as follows. Section II gives a brief overview about database replication strategies. Section III details the implementation of our experimental framework while the results of our experiments are presented in Section IV. Section V summarizes related work before we conclude the paper in Section VI.

\(^{4}\)http://aws.amazon.com/rds/
\(^{5}\)http://www.microsoft.com/windowsazure/sqlazure/

II. BACKGROUND

Data replication \([12]\) is a well-known strategy to achieve the availability, scalability and performance improvement goals in the data management world. In practice, two replication architectures are commonly used: the multi-master replication and the master-slave replication. On one side, the multi-master replication allows each replica to maintain a full copy of the database and to serve both of read and write transactions. In this architecture, write-write conflicts is resolved by the replication middleware, so that each replica executes write transactions in the same order. On the other side, the master-slave replication is more adequate for improving read throughput. In this architecture, read transactions are served by slaves while the all the write transactions are only served by the master. The replication middleware is in charge of passing writesets from the master to slaves in order to keep the database replicas up-to-date. The write-write conflicts can be all resolved on the master side. Our study focuses on the master-slave architecture as it is the most commonly employed architecture by many web applications.

In general, the replication architectures expose how read and write transactions are assigned across replicas while the synchronization models reveal how data is committed across replicas. For example, the synchronous replication blocks a successful response to the client until the write transaction is committed on the updated replica and writesets are duplicated over all other replicas. Therefore, it makes sure that all replicas are consistent during the time, however traversing all replicas potentially incurs high latency on write transactions. Furthermore, the availability of the system may be affected if unreachable replicas due to network partitioning cause suspension of synchronization. The asynchronous replication sends a successful response once the write transaction is committed on updated replica. The writesets will be propagated to other replicas at a later time. It avoids high write latency over networks in exchange of stale data. Moreover, once the updated replica goes offline before duplicating data, data loss may occur. In general, synchronous replication is essential to data sensitive applications (e.g. bank transactions). However, asynchronous replication is more favorable to many web applications (e.g. social network applications), which could be more tolerant with a wider window of data staleness (replication delay). Due to the existence of such a replication delay, read transactions on database replicas are not expected to return consistent results all the time. However, it is guaranteed that the database replicas will be eventually consistent \([8]\).

III. IMPLEMENTATION

A. Experiment design

The Cloudstone benchmark\(^7\) has been designed as a performance measurement tool for Web 2.0 applications. The benchmark mimics a Web 2.0 social events calendar that allows users to perform individual operations (e.g. browsing, searching and creating events), as well as, social operations

\(^{7}\)http://radlab.cs.berkeley.edu/wiki/Projects/Cloudstone
(e.g. joining and tagging events)[11]. Unlike Web 1.0 applications, Web 2.0 applications behave differently on database in many ways. One of the differences is on the write pattern. As contents of Web 2.0 applications depend on user contributions via blogs, photos, videos and tags. Therefore, more write transactions are expected to be processed. Another difference is on the tolerance with data consistency. In general, Web 2.0 applications are more acceptable to data staleness. For example, it might not be a mission-critical goal for a social network application (e.g. Facebook) to immediately have a user’s new status available to his friends. However, a consistency window of some seconds (or even some minutes) would be still acceptable. Therefore, we believe that the design and workload characteristics of the the Cloudstone benchmark is more suitable to the purpose of our study rather than other benchmarks such as TPC-W8 or RUBiS9 which are more representing Web 1.0-like applications.

The original software stack of Cloudstone consists of 3 components: web application, database, and load generator. Throughout the benchmark, the load generator generates load against the web application which in turn makes use of the database. The benchmark designs well for benchmarking performance of each tier for Web 2.0 applications. However, the original design of the benchmark limits the purpose of our experiments, which is mainly focusing on the database tier of the software stack, on the aspect that it is hard to push database to its performance limit. In general, a user’s operation which is sent by a load generator has to be interpreted as database transactions in the web tier based on a pre-defined business logic before bypassing the request to the database tier. Thus the saturation on the web tier usually happens earlier than the saturation on the database tier. Therefore, we modified the design of the original software stack by removing the web server tier. In particular, we re-implemented the business logic of the application in a way that a user’s operation can be processed directly at the database tier without any intermediate interpretation at the web server tier. Meanwhile, on top of cloud implementation, we also implemented a connection pool (i.e. DBCP10) and a proxy (i.e. MySQL Connector/J11) components. The pool component enables the application users to reuse the connections that have been released by other users who have completed their operations in order to save the overhead of creating a new connection for each operation. The proxy component works as a load balancer among the available database replicas where all write operations are sent to the master while all read operations are distributed among slaves.

Multiple MySQL replicas are deployed to compose the database tier. For the purpose of monitoring replication delay in MySQL, we have created a Heartbeats database and a time/date function for each replica. The Heartbeats database, synchronized in the format of SQL statement across replicas, maintains a 'heartbeat' table which records an id and a timestamp in each row. A heartbeat plug-in for Cloudstone is implemented to insert a new row with a global id and a local time stamp to the master periodically during the experiment. Once the insert query is replicated to slaves, every slave re-executes the query by committing the global id and its own local time stamp. The replication delay from the master to slaves is then calculated as the difference of two timestamps between the master and each slave. In practice, there are two challenges with respect to achieving a fine-grained measurement of replication delay: the resolution of the time/date function and the clock synchronization between the master and slaves. The time/date function offered by MySQL has a resolution of a second which represents an unacceptable solution because accurate measuring of the replication delay requires a higher precision. We, therefore, implemented a user defined time/date function with a microsecond resolution that is based on a proposed solution to MySQL Bug #852312.

With the customized Cloudstone14 and the heartbeat plug-in, we are able to achieve our goal on measuring the end-to-end database throughput and the replication delay. In particular, we defined two configurations of the read/write ratios: 50/50 and 80/20. We also defined three configurations of the geographical locations based on Availability Zones (they are distinct locations within a Region) and Regions (they are separated into geographic areas or countries): same zone, all slaves are deployed in the same Availability Zone of a Region of the master database; different zones, the slaves are in the same Region as the master database, but in different Availability Zones; different regions, all slaves are geographically distributed in a different Region from where the master database is located. The workload and the number of database replicas start with a small number and gradually increase at a fixed step. Both numbers stop increasing if there are no throughput gained.

B. Experiment setup

We conducted replication experiments in Amazon EC2. The Fig. 1 shows the experiment setup in Amazon EC2 service. The experiment setup is a three-layer implementation. The first layer is the Cloudstone benchmark which controls the read/write ratio and the workload by separately adjusting the number of read and write operations and the number of concurrent users. As a large number of concurrent users emulated by the benchmark could be very resource-consuming, the benchmark is deployed in a large instance to avoid any overload on the application tier. The second layer includes the

10http://commons.apache.org/dbcp/
11http://www.mysql.com/products/connector/
13http://www.ntp.org/
14the source code of our Cloudstone customized implementation is available on http://code.google.com/p/clouddb-replication/
master database that receives the write operations from the benchmark and is responsible for propagating the writesets to the slaves. The master database runs in a small instance so that saturation is expected to be observed early. Both of the master database server and the application benchmark are deployed in location of us-east-1a. The third layer is a group of slaves which are responsible for processing read operations and updating writesets. The number of slaves in a group varies from one to the number where throughput limitation is hit. Several options for the deployment locations of the slaves have been used, namely, the same zone as the master in us-east-1a, a different zone in us-east-1b and four possible different regions, ranging among us-west, eu-west, ap-southeast and ap-northeast. All slaves run in small instances for the same reason of provisioning the master instance.

Several sets of experiments have been implemented in order to investigate the end-to-end throughput and the replication delay. Each of these sets is designed to target a specific configuration regarding the geographical locations of the slave databases and the read/write ratio. Multiple runs are conducted by compounding different workloads and numbers of slaves. The benchmark is able to push the database system to a limit where no more throughput can be obtained by increasing the workload and the number of database replicas. Every run lasts 35 minutes, including 10-minute ramp-up, 20-minute steady stage and 5-minute ramp down. Moreover, for each run, both the master and slaves should start with a pre-loaded, fully-synchronized database.

IV. EXPERIMENTAL RESULTS

A. End-to-end throughput experiments

Fig. 2 and Fig. 3 show the throughput trends for up to 4 and 11 slaves with mixed configurations of three locations and two read/write ratios. Both experiment results indicate that MySQL with asynchronous master-slave replication is limited to scale due to the saturation happened to the master database. In particular, the throughput trends react to saturation movement and transition in database replicas in regards to an increasing workload and an increasing number of database replicas. In general, the observed saturation point (the point right after the observed maximum throughput of a number of slaves), appearing in slaves at the beginning, moves along with an increasing workload when more and more slaves are synchronized to the master. But eventually, the saturation will transit from slaves to the master where the scalability limit is achieved. Taking throughput trends with configurations of same zone and 50/50 ratio (Fig. 3a) as an example, the saturation point of 1 slave is initially observed under 100 workloads due to the fully utilization of the slave’s CPU. When a 2nd slave is attached, the saturation point shifts to 175 workloads where both slaves reach maximum CPU utilization while the master’s CPU usage rate is also approaching to its limitation. Thus, ever since the 3rd slave is added, 175 workloads remain as the saturation point, but with the master turns to be saturated instead of slaves. Once the master is in the saturation status, adding more slaves does not help with improving the scalability, because the overloaded master fails to offer extra capacity for improving write throughput to keep up the read/write ratio that corresponds to the increment of the read throughput. Hence, the read throughput is suppressed by the benchmark, for the purpose of maintaining pre-defined read/write ratio at 50/50. The slaves are over provisioned in the case of 3 and 4 slaves, as the suppressed read throughput prevents slaves from being fully utilized. The similar saturation transition also happens to 3 slaves at 50/50 ratio in other two locations (Fig. 2b and Fig. 2c), and 10 slaves at 20/80 ratio in same zone(Fig. 3a) and different zone(Fig. 3b) and also 9 slaves at 20/80 ratio in different region(Fig. 3c).

The configuration of the geographic locations is a factor that affects the end-to-end throughput, in the context of locations of users. In the case of our experiments, since all users emulated by Cloudstone send read operations from us-east-1a, distances between the users and the slaves increase, following the order of same zone, different zone and different region. Normally, a long distance incurs a slow round-trip time, which results in a small throughput for the same workload. Therefore, it is expected to observe a decrease of maximum throughput when configurations of locations follow the order of same zone, different zone and different region. Moreover, the throughput degradation is also related to read percentages, the higher percentage the larger degradation. It explains why degradation of maximum throughput is more significant with configuration of 80/20 read/write ratio (Fig. 3). Hence, it is a good strategy to distribute replicated slaves to places that are close to users to improve end-to-end throughput.

The performance variation of instances is another factor that needs to be considered when deploying database in the cloud. For throughput trends of 1 slave at 50/50 read/write ratio with configurations of different zone and different region, respectively, if the configuration of locations is the only factor, the maximum throughput in different zone(Fig. 2b) supposes to be larger than the one in different region(Fig. 2c). However, the main reason of throughput difference here is
caused by the performance variation of instances rather than the configuration of locations. The 1st slave from same zone runs on top of a physical machine with an Intel Xeon CPU E5430 2.66GHz. While another 1st slave from different zone is deployed in a physical machine powered by an Intel Xeon CPU E5507 2.27GHz. Because of the performance differences between physical CPUs, the slave from same zone performs better than the one from different zone. Previous research indicated that the coefficient of variation of CPU of small instances is 21% [13]. Therefore, it is a good strategy to validate instance performance before deploying applications into the cloud, as poor-performing instances are launched randomly and can largely affect application performance.

B. Replication delay experiments

1) Clock synchronization in cloud: Clock synchronization issue refers to the fact that internal clocks of physical machines may differ due to the initial clock setting and subsequent clock drift. It results in time differences between two machines even they are read at the same time. This issue could also happen to instances in the cloud environment, if two instances are deployed in distinct physical machines where the clock is not shared. As a matter of fact, it has been observed by Ristenpart et al. that all instances launched by a single Amazon EC2 account never run in the same physical node [14]. Hence, all running instances belong to a single account must have the clock synchronization issue.

Fig. 4 exposes how NTP synchronization keeps the time difference stable between two instances during 20-minute
Different zone (us-west-1b) | Different region (eu-west-1a)

Fig. 5: Average relative replication delay with an increasing workload, an increasing number of database replicas and different configurations to the geographical locations of the slave databases. The read/write ratio is 50/50. The initial data size is fixed at 300. The master is located in us-west-1a, while locations of slaves are noted in the sub-caption of each figure.

Average relative replication: 1) Average relative replication: 2) Average relative replication:

Fig. 6: Average relative replication delay with an increasing workload, an increasing number of database replicas and different configurations to the geographical locations of the slave databases. The read/write ratio is 80/20. The initial data size is fixed at 600. The master is located in us-west-1a, while locations of slaves are noted in the sub-caption of each figure.

period. If two instances only apply the NTP protocol once at the beginning of the experiment, the time difference between two instance surges linearly from 7 milliseconds up to 50 milliseconds. Its median is 28.23, and its standard deviation is 12.31. The surge is caused by clock drift, as Amazon synchronizes its instances in a very relaxed manner - every couple of hours. Thus, clock drift lies in between of two continuous time synchronization. If two instances apply the NTP protocol every second, then samples of all time differences mostly rest in between of 1 millisecond and 8 milliseconds. Its median is 3.30 ms, and standard deviation is 1.19. With time synchronization enabled every second, the time difference is more stable.

As explained in Section III-A, the replication delay in our experiments is measured based on committed local time stamps on two or more replicas. Thus, the clock synchronization issue also exists in the replication delay. As we are more interested in the changes of replication delay, rather than that of accuracy. We therefore can use average relative replication delay to eliminate the time differences introduced by the clock synchronization issue. The average relative replication delay is represented as the difference between two average replication delays on the same slave. One average replication delay represents the average of delays without running workloads while another represents the average of delays under a number of concurrent users. Both average is sampled with the top 5% and the bottom 5% data cut out as outliers, because of network fluctuation. As both average delays come with a stable time differences with NTP protocol enabled every second, the time difference can then be eliminated with a minus operation. In our experiments, we set the NTP protocol to synchronize with multiple time servers every second for a more stable time difference.

2) Average relative replication: Fig. 5 and Fig. 6 show the trends of the average relative replication delay for up to 4 and 11 slaves with mixed configurations of three locations and two read/write ratios. The results of both figures imply that the affect of the configurations of the geographical locations is less important than the workload characteristics on replication delay. The trends of the average relative replication delay respond to an increasing workload and an increasing number of database replicas. For most cases, with the number of database replicas kept constant, the average relative replication delay surges along with an increasing workload. Because an
increasing workload leads to more read and write operations sent to the slaves and the master database, respectively. It turns out; the increasing read operations result in a higher resource demand on every slave, while the increasing write operations on the master database leads to, indirectly, increasing resource demand on slaves as more writesets are propagated to be committed on slaves. The two increasing demands push resource contention higher, resulting in the delay of committing writesets, by which means increases replication delay. Similarly, the average relative replication delay decreases along with an increasing number of database replicas. Since adding a new slave leads to reducing the resource contention and hence decreasing the replication delay.

The configuration of the geographic location of the slaves play a less significant role in affecting replication delay, in comparison to the change of the workload characteristics. We measured 1/2 round-trip time between the master in us-west-1a and the slave that uses different configurations of geographic locations by running ping command every second for a 20-minute period. The results suggest an average of 16, 21, and 173 milliseconds 1/2 round-trip time for the same zone (Fig. 5a and Fig. 6a), different zones (Fig. 5b and Fig. 6b) and different regions (Fig. 5c and Fig. 6c), respectively. However, The trends of the average relative replication delay usually can go up to two to four orders of magnitude in Fig. 5, or one to three orders of magnitude in Fig. 6. Therefore, it could be suggested that the geographic replication would be applicable in the cloud as long as workload characteristics can be well managed (e.g. having a smart load balancer which is able of balancing the operations based on estimated processing time).

V. RELATED WORK

With the emergence of cloud computing, several studies have assessed the performance and scalability of cloud computing solutions for data management applications. Kossmann et al. [15] have used the TPC-W benchmark to evaluate the performance of different database architectures for processing OLTP workload in commercial database services (e.g. Amazon RDS, Amazon SimpleDB, Microsoft SQL Azure). The results of the experiments have shown that the the cost, performance and the scalability of the cloud services vary significantly depending on the characteristics of the workload (e.g. read/write ratio). AppScale [16] is an open source implementation of the Google App Engine (GAE) Datastore which unifies access to a wide range of open source distributed database technologies. AppScale has been used for conducting an evaluation of the performance characteristics of several NoSQL systems including: HBase, Hypertable, Cassandra, Voldemort and MongoDB. The Yahoo! Cloud Serving Benchmark (YCSB) [17] is another effort for benchmarking cloud serving systems. The benchmark consists of a package of workloads with different characteristics (e.g. read-heavy workloads, write-heavy workloads, scan workloads, etc). The initial implementation of the YCSB benchmark has been used for evaluating four systems: Cassandra, HBase, PNUTS, and a simple sharded MySQL in terms of their performance and elasticity characteristics. The scope of the benchmark has been recently extended, YCSB++ [18], to support complex features such as including multi-tester coordination for increased load and eventual consistency measurement, multi-phase workloads to quantify the consequences of work deferment and the benefits of anticipatory configuration optimization such as B-tree pre-splitting or bulk loading. The YCSB++ features are used for evaluating two BigTable-like table stores: Apache HBase and Accumulo.15

Florescu and Kossmann [19] argued for the importance of including consistency among the features that are measured, and they suggested that system evaluation should identify the tradeoff between data consistency and other properties such as performance or financial cost. Various studies have measured the performance impact of weaker consistency models which are used by different NoSQL systems and cloud services. Anderson et al. [20] have presented algorithms for analyzing the trace of interactions between the client machines and a key-value store. The algorithms can report whether the trace is safe, regular, or atomic, and if not, how many violations there are in the trace. Wada et al. [9] have experimentally investigated consumer-based observations of the consistency and performance properties of various offerings. The results of the study have shown that there is no observed benefit from accepting weaker consistency properties when using a single writer/reader. Bermbach and Tai [21] have evaluated Amazon S3 in terms of consistency guarantees when using multiple readers and found that S3 frequently violates monotonic read consistency. They also encountered strange periodicities which alternate approximately twice a day. Kraska et al. [22] have presented the consistency rationing mechanism that allows application developers to define the consistency guarantees on the data instead of at the transaction level and allows to automatically switch consistency guarantees at runtime. In particular, they presented a number of techniques that let the system dynamically adapt the consistency level by monitoring the data and gathering temporal statistics of the data.

Much research has investigated on the general characteristics of the wide spectrum of cloud computation platforms [13], [23], [24]. The CloudCmp project [23] acts as a systematic comparator of the performance and cost of cloud providers. It measures the elastic computing, persistent storage, and networking services offered by cloud providers (e.g. Amazon AWS, Microsoft Azure, Google AppEngine, and Rackspace CloudServers) using different metrics that reflect their impact on the performance of customer applications. The CARE framework [24] implements a unified interface with WSDL and REST in order to evaluate different Cloud platforms (e.g. Amazon EC2, Google App Engine, Azure) in terms of different characteristics such as: performance, availability and reliability. Schad et al. [13] have established microbenchmarks to measure performance variance in CPU, I/O, and network. They used a multi-node MapReduce application to quantify the impact on real data-intensive applications. The results of

15http://wiki.apache.org/incubator/AccumuloProposal
the study have shown that EC2 performance varies by a lot. They have related this variance to the different virtual system types provided by Amazon. In addition, they indicated that the choice of availability zone also influences the performance variability. Minhas et al. [25] have presented a study of the overhead of running a database workload on a virtual machine. The results of the study have shown that the average overhead is less than 10%.

VI. CONCLUSION

In practice, there are different approaches for deploying data-intensive applications in cloud platforms. In this paper, we have focused on the application-managed approach where the resources of the database tiers (database servers) are migrated to virtual machines in the public cloud. We have experimentally evaluated the behavior of the master-slave database replication strategy on Amazon EC2 using the Cloudstone benchmark and MySQL database. The experiments involved two configurations of the workload read/write ratio (50/50 and 80/20) and different configuration of the geographical locations of the database replicas.

The results of our study show that the performance variation of the dynamically allocated virtual machines is an inevitable issue that needs to be considered when deploying database in the cloud. Clearly, it affects the end-to-end throughput. Different configurations of geographic locations can noticeably affect the end-to-end throughput as well. For most cases, as the number of workload increases, the replication delay increases. However, as the number of slaves increases, the replication delay decreases. The effect of the configurations of geographic location is not as significant as increasing workloads in affecting the replication delay.

It should be noted that the results of our study can not be generalized and further investigations in different directions are left out for future work such as: hosting the database servers in EC2 instances with different sizes, using different database systems (e.g. SQL Server, Oracle, DB2) or using other infrastructure-as-a-service cloud providers for hosting the database virtual machines (e.g. Rackspace16, Joyent17). It would be also interesting to compare between the application-managed approach and the approach of relational database services (e.g. Amazon RDS) in terms of their performance characteristics (e.g. performance variance and replication delay) and their scalability limits.

VII. ACKNOWLEDGEMENT

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

This work is partially funded by AWS in Education Research Grant18.

REFERENCES


16http://www.rackspace.com/
17http://www.joyent.com/