Consistency in the Cloud: When Money Does Matter!

Houssem-Eddine Chihoub*, Shadi Ibrahim*, Gabriel Antoniu*, María S. Pérez†

*INRIA Rennes - Bretagne Atlantique
Rennes, 35000, France
{houssem-eddine.chihoub, shadi.ibrahim, gabriel.antoniu}@inria.fr

†Universidad Politécnica de Madrid
Madrid, Spain
mperez@fi.upm.es

***Abstract*—With the emergence of cloud computing, many organizations have moved their data to the cloud in order to provide scalable, reliable and highly available services. To meet ever growing user needs, these services mainly rely on geographically-distributed data replication to guarantee good performance and high availability. However, with replication, consistency comes into question.**

Service providers in the cloud have the freedom to select the level of consistency according to the access patterns exhibited by the applications. Most optimizations efforts then concentrate on how to provide adequate trade-offs between consistency guarantees and performance. However, as the monetary cost completely relies on the service providers, in this paper we argue that monetary cost should be taken into consideration when evaluating or selecting a consistency level in the cloud. Accordingly, we define a new metric called *consistency-cost efficiency*. Based on this metric, we present a simple, yet efficient economical consistency model, called *Bismar*, that adaptively tunes the consistency level at run-time in order to reduce the monetary cost while simultaneously maintaining a low fraction of stale reads. Experimental evaluations with the Cassandra cloud storage on a Grid'5000 testbed show the validity of the metric and demonstrate the effectiveness of the proposed consistency model.

***Keywords*-Cloud storage; geographical replications; consistency; Monetary cost; efficiency;**

I. Introduction

Cloud computing has recently emerged as a popular paradigm for harnessing a large number of commodity machines. In such a paradigm, users acquire computational and storage resources with respect to a pricing scheme similar to the economic exchanges in the utility market place: users can lease the resources they need in a Pay-As-You-Go manner [1]. For example, the Amazon Elastic Compute Cloud (EC2) is using a pricing scheme based on virtual machine (VM) hours: Amazon currently charges per small instance hour at $0.065 [2].

With data growing rapidly and applications becoming more data-intensive, many organizations have moved their data to the cloud aiming at providing scalable, reliable and highly available services. Cloud providers allow service providers to deploy and customize their environment in multiple physically separate data centers to meet the ever-growing user needs. Services therefore can replicate their state across geographically diverse sites and direct users to the closest or least loaded site.

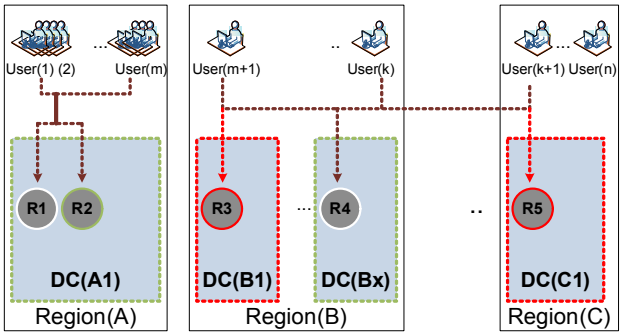


Figure 1. Leveraging Geographically-distributed data Replicas in the Cloud: *Having replications in different datacenters results with fast access (Users1 located at Region (A) and therefore directed to R1 and Users(m+1) located at Region(B) and therefore directed R3); Under heavy load — when multiple replicas coexists — the load will be shared by these replicas and therefore improve the performance (The heavy load distributed between R1 and R2 in Region); In case of failure, the load will be directed to the closest replica within the same datacenter if possible or on remote one (the replicas within DC(B1) and DC(C1) fails and the load is directed to DC(Bx))*

Replication has become an essential feature in storage systems and is extensively leveraged in cloud environments [3][4][5]. It is the main reason behind several features such as fast accesses, enhanced performance, and high availability.(as shown in Figure 1).

For **fast access** users' requests can be directed to the closest data center in order to avoid communications' delays and thus insure fast response time and low latency.

For **enhanced performance** users' requests can be re-directed to other replicas within the same data center (but different racks) in order to avoid overloading one single copy of the data and thus improve the performance under heavy load.

For **high availability** failure and network partitions are common in large-scale distributed systems; by replicating we can avoid single points of failure.

A particularly challenging issue that arises in the context of storage systems with geographically-distributed data replication is how to ensure a consistent state of all the replicas. Insuring strong consistency by means of synchronous replication introduces an important performance overhead due to the high latencies of networks across data centers

(the average round trip latency in Amazon sites varies from 0.3ms in the same site to 380ms in different sites [6]). These high latencies may generate significant financial losses for service providers that use such storage systems. For instance, the cost of a single hour of downtime for a system doing credit card sales authorizations has been estimated to be between 2.2M$-3.1M$ [7]. Consequently, many Internet services tend to rely on storage systems with eventual consistency. Eventual consistency allows the system to return some stale data at some points in time, but ensures that all data will eventually become consistent.

Recently many cloud storage systems have been developed, such as Dynamo [8], Cassandra [9], BigTable [4], Yahoo! PNUTS [10], and HBase [11]. These solutions are practical to use as cloud and web service storage backends. They allow many web services to scale up their systems in an extreme way, while maintaining performance with very high availability. For example, Facebook uses Cassandra to scale up to host data for more than 800 million active users [12]. However, the undoubted availability and performance of such solutions prove to be too costly in terms of inconsistency. As shown in [13], under heavy reads and writes some of these systems may return up to 66.61% stale reads. This is an alarming rate, as it means that most probably two out of three reads are useless. This in turn adversary impacts the financial profit of the services providers: it generates significant costs as it violates the SLAs of services users.

Consistency-performance and consistency-availability trade-offs have long been investigated in literature: many consistency optimization solutions have been devoted to improving the application's throughput and/or latency while preserving acceptable stale reads rate.However, in the area of cloud computing, the economical cost of using the rented resources is very important and should be considered when choosing the consistency policy.

To address these issues, this paper makes the following three contributions:

1) **Service/bill details.** To our knowledge, this is the first study to provide in-depth understanding of the monetary cost of cloud services with respect to their adopted consistency models. We discuss the different resources contributed to a service and the cost of these resources. This paper introduces an accurate decomposition of the total bill of the services into three parts with respect to the contributed resources:
   - *Instances cost:* the cost of leasing the virtual machines' instances during the running time of the service.
   - *Storage cost:* the cost of the storage space attached to the virtual machine and the cost of the requests as well.
   - *Network cost:* the cost of the network traffic including the inter- and intra-datacenter communications.

   To complement our analysis, a series of experiments are conducted to measure the monetary cost of different consistency levels in the Cassandra system [9] on Grid'5000 [14] and Amazon EC2 [2]. Such a study is important, as a big-picture understanding of the consistency in geo-replicated systems must take into account the monetary cost within the cloud.
2) **Novel metric.** We define a new metric called consistency-cost efficiency to evaluate consistency in the cloud.
3) **Equitable consistency and low cost.** Based on our metric, we introduce a simple yet efficient approach named *Bismar*, which adaptively tunes the consistency level at run-time in order to reduce the monetary cost while simultaneously maintaining a low fraction of stale reads. *Bismar* relies on a consistency probabilistic model that estimates the stale reads and the relative costs of the application according to the current read/write rates and network latency.

We have implemented *Bismar* with intensive evaluations on the Cassandra cloud storage system on Grid'5000 [14]. We use the Yahoo! Cloud Serving Benchmark (YCSB) [15] to mimic a real cloud serving environment with elastic access pattern workloads. We show that *Bismar* can lead to efficient costs without exceeding the tolerated number of stale reads on the applications.

Our paper is the first to provide a thorough analysis of the consistency cost in cloud storage systems. We view our paper as a necessary step for bridging the gap between the business model of the cloud and the research community in distributed systems aiming at designing and building more efficient and economical-oriented consistency models for cloud services.

This paper is organized as follows. Section II briefly discusses the eventual consistency model in the cloud and focuses on the billing details when adopting this model in the cloud. Section III discusses the different resources contributed to service and the cost of these resources, and reports on our empirical study on the monetary cost of consistency models. Then section IV describes the new cost efficiency metric for consistency in the cloud. In section V we describe the *Bismar* implementation and present detailed results of experimental evaluations. Section VI discusses related work. Finally, section VII presents our conclusions and future work.

II. Background and motivation

A. Eventual Consistency in the Cloud

The way consistency is handled has a big impact on the performance. Traditional synchronous replication (strong consistency) dictates that an update must be propagated to all the replicas before returning success. In cloud services

where data updates occur often, it is difficult to keep the consistency among replicas across the entire cloud storage system. To solve this problem, eventual consistency with an asynchronous quorum replication has been introduced. Here the consistency level is chosen on a per-operation basis and is represented by the number of replicas in the quorum (a subset of all the replicas). Data accesses and updates are performed to all replicas in the quorum. Thus, using this level for both read operations and write operations guarantees that the intersection of replicas involved in both operations contains at least one replica with the last update.

Many cloud storage systems such as Dynamo [8], Cassandra [16], Voledemort [17], and Riak [18] adopt asynchronous quorum replication [19][20]. This gives the application writer more flexibility when selecting the type of consistency that is appropriate for each operation.

B. Service's Bill Decomposition

Since cloud computing is an economically-driven distributed system paradigm, deploying and running services and applications in the cloud comes with monthly bill. In general, services require a set of linked servers (distributed in multi-sites) to run the web-services' applications; these servers are attached to a group of storage devices which store the services' data. With respect to cloud resources' offers, a basic service bill will include charges for the following resources[1]:

Computing resources. Virtual machines equipped with a certain amount of CPU and memory resources. Cloud IaaS providers offer different VM instances — varying in the resource's capacity and accordingly the prices — and typically charged for the incurred virtual machine hours. For example, Amazon EC2 [2] offers a set of instances with different configurations and prices: while the cheapest instance (small instance, equivalent to a server with a CPU capacity of a 1.0-1.2GHz and memory size of 1.7GiB) comes at cost of $0.065\$$ per Hour, the most expensive instance (High I/O Quadruple Extra Large Instance, equivalent to a server with CPU capacity of $35 \times 1.0 - 1.2$GHz and memory size of 60.5GiB) comes at cost of $3.100\$$ per Hour.

Storage resources. Cloud IaaS providers offer two types of storage services that are different in their pricing and usability. Taking Amazon Web Services as an illustrating example, there are two representative storage services: Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Block Store (Amazon EBS). The storage services are typically billed according to the used GBs per month and number of requests to the stored data. Taking into account the tremendous amount of data that current services need to manage and maintain, and the need to reduce the latency of data movement when processing data, Amazon EBS becomes the customer's first choice to achieve not only highly scalable and high performance services but highly reliable and predictable ones as well. This is despite the fact that Amazon EBS can be attached to any running Amazon EC2 instance and can be exposed as a device within the instance. Consequently, in this study we adapted the Amazon EBS pricing scheme.

Network resources. Cloud IaaS providers equip their infrastructure with high speed networks not only within data centers but also across geographically distributed centers. This comes at a monetary cost, although services don't currently reflect the network usage and cost. The network cost is usually embedded within the cost of other services (computational service and storage services), and it varies in accordance to the service type and within/across sites (e.g., the cost of data transfer between Amazon EC2 instances is zero if they are located in the same availability zone).

C. Monetary Cost of consistency: Why it does matter?

With data growing rapidly and applications becoming more data-intensive, a large class of organizations have migrated their data along with their storage backends into the cloud as to provide efficient services in terms of scalability, reliability, and availability : Cloud providers allow service providers to deploy and customize their environment in multiple physically separated data centers to meet the ever-growing users' needs. Services therefore can replicate their date across geographically diverse sites and direct users to an appropriate site based on the locality of access and site load. Thus, replication has become a necessity in these services. However, with replication, consistency comes into question.

We observe that stronger consistency by the means of synchronous replications may introduce high latencies due to the cross-sites communication and therefore will significantly increase the monetary cost of the services:

1) High latency causes high monetary cost. Obviously because the cost of leasing a VM-instance is proportional to the latency (run-time), in addition to the increased cost of both the storage (e.g., number of requests to the copies) and the communication cost (e.g., number of cross-sites communication) due to the synchronous cross-site replication.
2) High latency causes significant financial losses for service providers that use such storage systems. For instance, the cost of a single hour of downtime for a system doing credit card sales authorizations has been estimated to be between 2.2M$-3.1M$ [21].

On the other hand, we observe that eventual consistency or weaker consistency may reduce the monetary cost with respect to a lower maintained latency and therefore lower instance costs, but this comes at the risk of increasing the rate of stale data (e.g., [13] demonstrated that under heavy reads and writes some of these systems may return up to

[1]The pricing of some cloud services (computing and storage services) may vary at different providers or at different provider-sites. As the goal of our study is to explore the consistency cost variation, we assume that the computing and storage pricing is the same at different sites

66.61% stale reads). This in turn adversarily impacts the financial profit of the service providers: it generates significant financial losses as it violates the SLAs of services users. This makes eventual consistency a two-edged sword. While the eventual consistency has been exploited extensively in literature and commercial products, its monetary cost and negative impacts on the stale reads rate have been largely ignored.

The *aforementioned observations*, combined with the urgent need to address the consistency-cost efficiency and stale reads problems associated with quorum replications, motivate us to an in-depth study of the monetary cost of the different consistency levels in the cloud and — as a result — to propose our cost efficient optimization.

III. Microscopic of Consistency Cost

In section 2.2, we gave a big-picture understanding of the cost of services deployed in the cloud by describing the different resources contributed to obtain a certain level of consistency in geo-replicated storage systems. In this section we complement our macroscopic analysis with a detailed analysis of the consistency cost in the cloud, using a widely used open source geo-replicated storage system that supports multi-level consistency as an illustrated example, namely Cassandra [9].

Ideally, we would like to get a deep idea of why different consistency levels may result in different costs, how the resources accordingly contribute to the total cost, and how background operations such as read repair can impact the overall cost.

The choice of consistency level cl affects all of these three costs. When higher consistency levels are required more replicas are involved in the requests. That affects both operations latency and throughput, which leads to higher runtime. Similarly, network traffic grows higher with higher consistency levels, which leads to a higher networking bill. Moreover, higher consistency levels generate a higher number of requests from storage devices, directly affecting storage cost.

Formula 1 presents the overall cost for geo-replicated based services for a given consistency level cl. Essentially, this cost is the combination of the VM instances cost $\text{Cost}_{in}(\text{cl})$, the backend storage cost $\text{Cost}_{st}(\text{cl})$, and network cost Cost_{tr} (cl).

$$\text{Cost}_{all}(\text{cl}) = \text{Cost}_{in}(\text{cl}) + \text{Cost}_{tr}(\text{cl}) + \text{Cost}_{st}(\text{cl}) \quad (1)$$

A. Computing unit: Instances cost

A common pricing scheme used by recent cloud providers is primarily based on virtual machine (VM) hours. Formula 2 presents the cost of leasing nbInstances VM-instances for a certain time runtime.

$$\text{Cost}_{in}(\text{cl}) = \text{nbInstances} \times \text{price} \times \lceil \frac{\text{runtime}}{\text{timeUnit}} \rceil \quad (2)$$

Here the price is the dollar cost per timeUnit[2] (e.g., In Amazon EC2 small instance the price is 0.065 per $hour$).

In order to generalize our pricing model and avoid inaccurate pricing due to unexpected network behavior (especially that we are studying the consistency cost in geo-distributed sites), we present the runtime in the form of number of operations $nbOps$ in the workload while fixing the throughput of a specific consistency level.

$$\text{runtime} = \frac{\text{nbOps}}{\text{throughput}} \quad (3)$$

The throughput varies from one consistency level to another according to the size of the internal traffic between sites.

B. Storage cost

As mentioned earlier the storage cost includes the cost of leased storage volume (GB per month) and the cost of I/O requests to/from this attached storage volume. In Amazon EC2 for instance, this would be the cost of attaching Amazon EBS to VM-instances in order to increase the storage capacity using a highly durable and reliable way. The total storage cost is accordingly given by Formula 4:

$$\text{Cost}_{st}(\text{cl}) = \text{costPhysicalHosting} + \text{costIORequests} \quad (4)$$

Based on the size of hosted data (including all data replications) nbNodes $\times$ dataSize where dataSize is the average data size per volume attached to VM-instance (locality and load balancing are important features in current data centers), we calculate the $costPhysicalHosting$ in Formula 5.

$$\text{costPhysicalHosting} = \text{nbNodes} \times \lceil \frac{\text{dataSize}}{\text{sizeUnit}} \rceil \times \text{price} \quad (5)$$

where the price is the dollar cost per sizeUnit (e.g., In Amazon EBS the price is 0.10 per $GB - month$).

We further estimate costIORequests in Formula 6.

$$\text{costIORequests} = \frac{\text{cl} \times \text{nbOps} + \text{readRepairIO}}{\text{nbRequestsUnit}} \times \text{price} \quad (6)$$

where nbOps is the number of operation with respect to the consistency level cl (it varies according to the number of replications involved in an operation). The read repair in a background operation is mostly triggered when inconsistency is detected. It generates requests to the storage devices and therefore it is important to include the read repair operations in our formula readRepairIO (more details about the read repair function will be provided in next section 3.1).

[2]We use the ceiling function because most providers charge each partial instance-hour as a full hour.

C. Network cost

Network cost varies in accordance to the service type of the source and destination (e.g., computational service and storage services) and whether the data transfer is within or across sites. In general, inter-datacenter communications is more expensive than intra-datacenter communications. Formula 7 shows the total cost of network communications as the sum of inter- and intra-data center communications [3](trafficInterDC and trafficIntraDC).

$$\begin{aligned}\text{Cost}_{tr}\ (\text{cl}) = \text{price(interDC)} \times \lceil \frac{\text{trafficInterDC}}{\text{sizeUnit}} \rceil \\ + \text{price(intraDC)} \times \lceil \frac{\text{trafficIntraDC}}{\text{sizeUnit}} \rceil\end{aligned} \quad (7)$$

where price(interDC) and price(intraDC) are the dollar cost per sizeUnit.

Hereafter we illustrate how to estimate both the inter- and intra-datacenter traffic.

Formula 8 shows our model of the inter datacenter, trafficInterDC, given the replicas communication interDcRep, the request routing $requestrouting$, and the internal mechanisms traffic IMechTraffic.

$$\begin{aligned}\text{trafficInterDC} = \text{interDcRep} + \text{requestRouting} \\ + \text{IMechTraffic}\end{aligned} \quad (8)$$

The inter-site traffic generated by the replicas communications strongly depends on the consistency level and the distribution of replication among datacenters (i.e., the number of replicas involved in a request to other datacenters which can be estimated as $\lfloor (\text{nbDc - 1}) \times \frac{\text{cl}}{\text{nbDc}} \rfloor$[4] where nbDc is the number of datacenters). Formula 9 shows our estimation of the inter traffic generated by the replicas communications.

$$\begin{aligned}\text{InterDcRep} = \lfloor (\text{nbDc - 1}) \times \frac{\text{cl}}{\text{nbDc}} \rfloor \\ \times \text{AvgDataSize} \times \text{nbOps}\end{aligned} \quad (9)$$

where avgDataSize is the average data size needed to be propagated to other replicas for one operation.

The traffic generated by the request routing and internal mechanisms depends essentially on the storage system design and implementation. Since our approach is destined to run on Cassandra storage, hereafter we illustrate such values with respect to this particular storage system. In Cassandra, all nodes (peers) have equal ranges of data and thus have an equal number of keys: this implies that each node is responsible for $\frac{1}{number\ of\ nodes}$ fraction of the keys.

[3]For simplicity, we consider only two geographical areas within which the prices differ. Some cloud providers may have more geographically-oriented prices: within availably zone, within regions, between regions. However, our pricing model can be easily extended to any number of geographical-oriented pricing options.

[4]For example if the (nbDC=3) and number of replicas involved in an operation (cl=4), the estimated number of replicas involved in a request on other dataceters is $\lfloor 2 \times \frac{4}{3} \rfloor = \lfloor \frac{8}{3} \rfloor = 2$ where $\lfloor\ \rfloor$ is a floor function.

Giving the number of nodes as nbNodes and the average number of nodes per datacenter avgNodesDc, the average number of request routing for an operation can be estimated as $\frac{\text{nbNodes} - \text{avgNodesDc}}{\text{nbNodes}}$. The size of inter traffic generated by request routing for a number of operations nbOps is therefore denoted as Formula 10.

$$\begin{aligned}\text{requestRouting(interDC)} = \frac{\text{nbNodes} - \text{avgNodesDc}}{\text{nbNodes}} \\ \times \text{nbOps} \times \text{avgDataSize}\end{aligned} \quad (10)$$

In Cassandra storage, the main internal traffic is generated by the gossip traffic and read repair mechanism as shown in Formula 11. The gossip traffic — used to share the state of nodes in the ring — is relatively small since it is just transmitting the state of one node which is negligible compared to data transfer.

$$\begin{aligned}\text{IMechTraffic} = \text{gossip(interDc)} \\ + \text{readRepair(interDc)}\end{aligned} \quad (11)$$

On the other hand, the read repair is used to propagate data to out of date (stale) replicas. The read repair function is triggered in two cases:

1) At random times for some requests: defined by the system administrator.
2) Whenever inconsistency is detected.

Formula 12 shows that read repair traffic depends on the probability or chance of triggering the mechanism *rrChance* which is defined by the storage administrator, as well as the chance of detecting mismatching replicas' timestamps *mmChance* $= \frac{\text{rf} - \text{cl}}{\text{rf}} \times \frac{\text{nbWrites}}{\text{nbReads} + \text{nbWrites}}$, where rf is the replication factor, nbWrites and nbReads are the number of write and reads.

$$\begin{aligned}\text{readRepair(interDC)} = \text{nbOps} \times \text{avgDataSize} \\ \times (\text{rrChance} \times \lfloor \frac{\text{rf}}{\text{nbDc}} \rfloor + \text{mmChance} \times \lfloor \frac{\text{rf} - \text{cl}}{\text{nbDc}} \rfloor)\end{aligned} \quad (12)$$

Computing the intra datacenter traffic size is very similar to the one of inter datacenter traffic. However, the intra traffic size of request routing is given by Formula 15.

$$\begin{aligned}\text{requestRouting(intraDC)} = \\ \frac{\text{avgNodesDc} - 1}{\text{nbNodes}} \times \text{nbOps} \times \text{avgDataSize}\end{aligned} \quad (13)$$

Similarly, we only consider the traffic in-between replicas within the same datacenter: Accordingly, the intra-site traffic generated by the replicas communications is denoted as in Formula14 and the read repair traffic is given by Formula 15

$$\text{intraDcRep} = (\lceil \frac{\text{cl}}{\text{nbDc}} \rceil - 1) \times \text{avgDataSize} \times \text{nbOps} \quad (14)$$

$$\text{readRepair(intraDC)} = \text{nbOps} \times \text{avgDataSize} \times (\text{rrChance} \times (\text{rf} - \lfloor \frac{\text{rf}}{\text{nbDc}} \rfloor) + \frac{\text{rf} - \text{cl}}{\text{rf}} \times ((\text{rf} - \text{cl}) - \lfloor \frac{\text{rf} - \text{cl}}{\text{nbDc}} \rfloor)) \quad (15)$$

D. Practical View of Consistency Cost in Cassandra

As we mentioned, our goal is to investigate the monetary cost variation of geo-replicated storage systems when adopting different consistency levels. Consequently we complement and benefit our earlier analyze, by evaluating the monetary cost in Cassandra.

1) Experimental setup: We run our experiments on Grid'5000 [14] and Amazon Elastic Compute Cloud (EC2). On Grid'5000, we deployed Cassandra on two datacenters (sites): with 30 nodes on the *Sophia* site and 20 nodes on the *Nancy* site as shown in Figure 2(a). All the nodes in *Sophia* are equipped with a 250GB hard disk, 4GB of Memory, and 4-cores AMD Opteron. The nodes in Nancy are equipped with disks of 320GB space, 16GB of Memory, and 8-cores Intel Xeon. The network connection between the two sites is provided by RENATER (The French national telecommunication network for technology, education, and research). It consists of a standard architecture of 10Gbit/s dark fibers. The network route between the two sites is the following: *Nancy-Paris-Lyon-Marseille-Sophia*: the average round trip latency is on average 0.230ms within the same site and 18.2ms in-between the two sites. On Amazon EC2, we also deployed Cassandra on 18 large instances (m1.large) on two availability zones: 10 instances on *us-east-1a* and 8 instances on *us-east-1d*. The average round trip latency is on average 0.284ms within the same site and 0.813ms in-between the two availability zones.

We used Cassandra-1.0.2 with a replication factor of 5 replicas: 2 replicas are allocated in *Nancy* and 3 replicas in *Sophia* (The same replication factor is used in Amazon EC2: 2 replicas in *us-east-1d* and 3 replicas in *us-east-1a*). Our replication strategy uses *NetworkTopologyStrategy* to enforce replication across multiple datacenters. We adopt the pricing schemes from Amazon web services as shown in Table 1[5]. We study the cost variation by evaluating different consistency levels (e.g., eventual consistency: one, two, Quorum: three, and strong consistency: All).

2) Micro Benchmark: We aim at a micro benchmark representing typical workloads in current services hosted in clouds. Based on case studies [10][22], we have selected the Yahoo! Cloud Serving Benchmark (YCSB) framework [15]. YCSB is used to benchmark Yahoo! cloud storage system "PNUTS" [10]. It is extended to be used with a variety of *open-source* data stores such as mongoDB [23] , Hadoop HBase [11] and Cassandra [16]. YCSB provides the features of a real cloud serving environment such as scale-out, elasticity and high availability. For this purpose, several workloads have already been proposed in order to apply a heavy read load, heavy update load, and read latest load, among other workloads. Also, the benchmark is designed to make the integration of new workloads very easy.

[5]The price of Amazon EC2 large instance was $0.32 at the time of writing this paper and it is now $0.26. However, as this price is applied to all consistency levels the difference in the pricing therefore doesn't affect our results and findings.

Table I
PRICING SCHEMES USED IN OUR EVALUATION

Computing unit *Large instance*	Storage unit	Storage Requests	Intra comm	Inter Comm
0.32$ per hour	0.10$ per GB/month	0.10$ per 1 million Requests	0.00$ per GB	0.01$ per GB

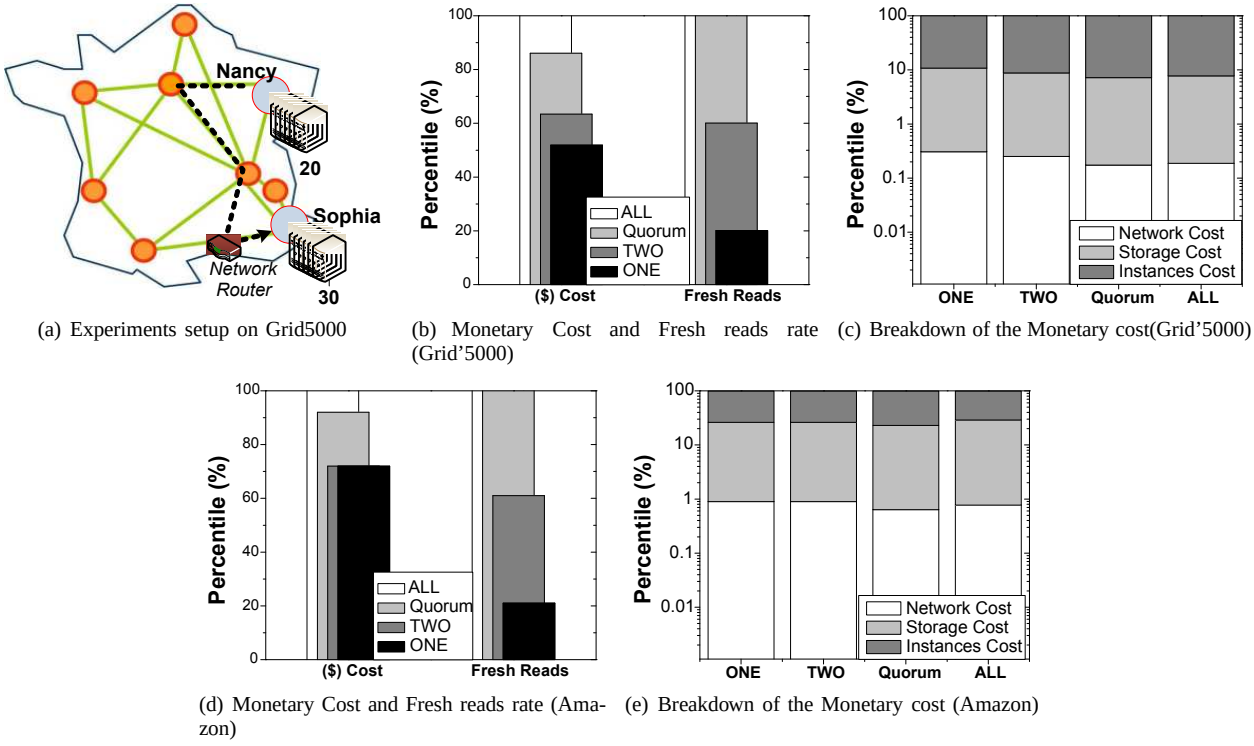
We use YCSB-0.1.3 and we run WorkloadA which is a heavy read-update workload (read/update ratio: 60/40). In both environments, our workload consists of 10 million operations on 5 million rows with a total of 23.84GB of data after replication.

3) Results on Grid'5000: As shown in Figure 2(b), the total monetary cost decreases when degrading the consistency level: the cost reduces from $138.76 — when the consistency level is set to *ALL* — to $71.72 when the consistency level is *ONE* (i.e., weak consistency reduces the cost by almost 48%). This result was expected as lower consistency levels involve fewer replicas in the operations, and thus maintaining low latency, less I/O requests to the storage devices, and less network traffic in general (the runtime of WorkloadA varies from 4 hours to 7 hours according the consistency level). This cost reduction, however, comes at the cost of a significant increase in the stale reads rate: as shown in Figure 2(b) 79% of the reads are stale reads — only 21% of the reads are fresh reads — when the consistency level is set to *ONE*.

Furthermore, it is obvious that degrading the consistency level for Quorum (here the number of replicas involved in an operation is 3 replicas) reduces the total cost by 13% while maintaining a zero stale reads rate as shown in Figure 2(b). This is because the storage system answers the read requests with the most up-to-date replica (fresh reads), which is always in the replicas quorum. Moreover, degrading the consistency level to TWO reduces the total monetary cost by almost 36%, but it adversary impacts the system consistency: only 61% of the reads where fresh reads.

Observation 1: The total cost of geo-replicated services strongly depends on the consistency level adopted: stronger consistency has higher cost but higher rate of fresh reads and vice versa. However, as services differ in their tolerable stale reads and their access pattern (within the same service: there is a significant diurnal variation in the access patten and the load levels). There is a need to define new metric to define the consistency level of an application.

Figure 2(c) shows the breakdown of the total cost accord-



(a) Experiments setup on Grid5000

(b) Monetary Cost and Fresh reads rate (Grid'5000)

(c) Breakdown of the Monetary cost(Grid'5000)

(d) Monetary Cost and Fresh reads rate (Amazon)

(e) Breakdown of the Monetary cost (Amazon)

Figure 2. Experiments setup and results on *Grid'5000* and Amazon EC2

ing to the contributed resources. In general, the instances cost has the higher cost amongst other resources (storage and network): it contributes to almost 90% of the service bill while the storage and network contribute on average to only 9% and 0.4%, respectively. This is due to our experiments' scale — number of operations — and the cheap prices of resources (as shown in Table I the intra communication is free of charges).

As shown in Figure 2(c), *storage cost* has a relatively lower contribution to the total cost for stronger consistency (ALL and Quorum) compared to weaker consistency (ONE and TWO): it contributes on average to 7.2% for the stronger one and 9% for the smaller one. The ALL consistency level requires higher nbOps compared to Quorum while both have zero/low readRepairIO and thus according to Formula (6) ALL has a relatively higher storage cost contribution in contrast to Quorum (e.g., it is 7% for Quorum and 7.5% for ALL). Moreover, although the nbOps is smaller for (ONE and TWO) compared to (ALL and Quorum) but the increasing number of readRepairIO increases the storage cost. Furthermore, as the cost of readRepairIO is proportional to the rate of stale reads, ONE has higher storage cost contribution in contrast to TWO.

In summary, the read repair function — ensuring that all outdated replicas become up to date —plays a very important role in determining the cost of storage with different consistency levels.

Network cost has also relatively a lower contribution to the total cost for stronger consistency (ALL and Quorum) compared to weaker consistency (ONE and TWO): it contributes on average to 0.175% for the stronger one and 0.275% for the smaller one. The ALL consistency level requires higher interDcRep compared to Quorum (higher number of involved replicas as well as Quorum always tends to answer the requests by involving the most close replicas "within the same data center if possible") while both have zero/low IMechTraffic and thus according to Formula 11 ALL has a relatively higher network cost contribution in contrast to Quorum. Moreover, although the interDcRep is smaller in for (ONE and TWO) compared to (ALL and Quorum) but the increasing size of IMechTraffic — due to the high rate of stale reads — increases the network cost. Furthermore, as the cost of IMechTraffic is proportional to the rate of stale reads, ONE has higher storage cost contribution in contrast to TWO.

Observation 2: Higher consistency causes a higher contribution to instances cost due to the high latency, and it causes a relatively lower contribution to both the storage and network cost as it avoids the extra cost caused by the read repair function.

4) Results on AmazonEC2: Figures 2(d) and 2(e) support our earlier findings and observations with Grid'5000. The total cost variation in Amazon is lower than in Grid'5000, because of the more powerful machines and the lower cross-

sites latency.

As shown in Figure 2(d), the total monetary cost decreases when degrading the consistency level: the cost reduces from \$32.39 — when the consistency level is set to *ALL* — to \$23.23 when the consistency level is *ONE* (i.e., weak consistency reduces the cost by almost 28%). This result was expected as lower consistency level involves fewer replicas in the operations, and thus maintaining low latency, less I/O requests to the storage devices, and less network traffic in general (the run-time of WorkloadA varies from 2 hours to 3 hours and 33 minutes according to the consistency level). This cost reduction, however, comes at the cost of a significant increase in the stale reads rate: as shown in Figure 2(d) 79% of the reads are stale reads — only 21% of the reads are fresh reads — when the consistency level is set to *ONE*.

Moreover, the costs of the ONE and TWO levels are the same, although there were significant variations in the running time (2hours and 1minutes for ONE and 2hours and 33minutes for TWO) and also significant variations in the network traffic and storage requests. This is because of the coarse-grained pricing units (per instance hour and per GB storage and per 1 million operations, etc).

Figure 2(e) shows the breakdown of the total cost according to the contributed resources. The instances cost has the higher cost amongst other resources (storage and network): it contributes to almost 74% of the service bill while the storage and network contribute on average to only 25.2% and 0.8%, respectively. This is due to our experiments' scale — number of operations — and the cheap prices of resources (as shown in Table I the intra communication is free of charges). Moreover, the ratio of the cost of the instances, storage and network to the total cost in Amazon EC2 is different from Grid'5000, because the shorter running time (the high throughput and the powerful machines) which in turn makes the instances cost smaller compared to other resources.

As shown in Figure 2(e), ALL has a relatively higher storage cost contribution in contrast to Quorum (e.g., it is 28% for ALL and 22% for Quorum). This is because the ALL consistency level requires higher nbOps compared to Quorum while both have zero/low readRepairIO. Moreover, although the nbOps is smaller for (ONE and TWO) compared to (ALL and Quorum) but the increasing number of readRepairIO increases the storage cost. Furthermore, as the cost of readRepairIO is proportional to the rate of stale reads, ONE has higher storage cost contribution in contrast to TWO. The *Network cost* has also relatively a lower contribution to the total cost for stronger consistency (ALL and Quorum) compared to weaker consistency (ONE and TWO): it contributes on average to 0.7% for the stronger one and 0.9% for the smaller one. This cost varies from one consistency level to another according to the number of involved replicas for the stronger consistencies and number of stale reads for the weaker ones.

IV. Consistency-Cost Efficiency Metric

As discussed in earlier, data consistency can strongly impact the financial cost of a certain service (i.e., while stronger consistency with high latency implies higher monetary cost of operation as demonstrated in section 4, the weaker consistency with high throughput causes higher operational cost because of the high rate of stale rate). Consequently, monetary cost should be considered when evaluating the consistency in the cloud [24].

As cloud computing is an economy-driven distributed system where monetary cost is explicate and measurable metric [25], we argue that the consistency-cost trade-off can be easily exposed in the cloud. Therefore in this paper, we define a new metric — consistency-cost efficiency — that exposes the tight relation between the degree of achieved consistency for a given monetary cost. Our goal is to define general yet accurate metric to evaluate consistency and thus using this metric as an optimization metric for cloud systems. Accordingly we define the consistency-cost efficiency as the ratio of consistency, measured by the rate of fresh reads, to the relative consistency cost as shown in Formula 16.

$$\text{Consistency-Cost Efficiency} = \frac{\text{Consistency}(cl)}{\text{Cost}_{rel}(cl)} \quad (16)$$

Where $Consistency(cl) = 1 - stalereadsrate$ and Cost_{rel} is the relative consistency cost with respect to the strong consistency and given by Formula 17.

$$\text{Cost}_{rel}(cl) = \frac{\text{Cost}(cl)}{\text{Cost}(cl\_all)} \quad (17)$$

It is important to mention that our metric is designed and can only be applied when strong consistency is not required by an application: we can consider our metric as system optimization for eventual consistency (i.e., tune the consistency to reduce the monetary cost without violating the application's requirements of fresh read rate).

V. Economical Consistency Approach

A. Design

We design and implement our approach with the following goals:

Extendable consistency-cost efficiency. Our solution aims at providing consistency guarantees while reducing the monetary cost. Therefore, we propose to use the consistency-cost efficiency (described in section IV) as an optimization metric: simply by selecting the consistency level with maximum consistency-cost efficiency. Moreover, to meet the diversity of applications requirements (e.g., cost constraint and fresh reads rate constraint), our solution can be easily extended to enable consistency-cost efficiency while favoring either cost or consistency.

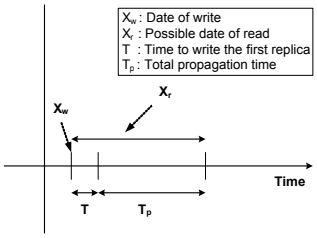


Figure 3. Situation that leads to a stale read

Self-adaptive. With the ever growing diversity in the access patterns of cloud applications along with the unpredictable diurnal/monthly changes in services loads, it is important to provide a self-adaptive approach that transparently scales the consistency level up/down at runtime without any human interaction. Our approach, therefore, embraces an estimation model for consistency-cost efficiency that could be achieved with different consistency levels: at runtime, the application's access pattern and network latency are fed to the consistency probabilistic estimation model (we have extended the model in [26] as will be explained later in this section) in order to estimate the rate of stale data that could be read in the storage system. Furthermore, these data along with information about pricing units in the targeted cloud platform are used to compute the monetary cost.

Pricing independent. Our solution targets public cloud and is not limited to any cloud provider in terms of provided services or pricing schemes. The fine-grained monetary cost analysis that is used for cost estimation (introduced in Section *III*) can be easily adopted to different services and pricing.

cloud storage systems independent. Since our solution is implemented as a separate layer at the top of the cloud storage system, it does not impose any modifications to the cloud system code. Our approach, therefore, can be applied to different cloud storage systems that are featured with flexible consistency rules.

1) Consistency Probabilistic Estimation: In our previous work [26], we propose an estimation of the stale read rate in the system by means of probabilistic computations. We define the situation that leads to a stale read in Figure 3. The read may be stale if its starting time X_w is in the time interval between the starting time of the last write and the end of the propagation time of data to the other replicas. This situation is repeatable for any write dates that may occur in the system. T_p in Figure 3 is the time necessary for the propagation of a write or an update to all the replicas. It is computed based on the network latency L_n and the average write size avg_w and should be represented as $T_p(L_n, avg_w)$, but in order to simplify the representation, it will be denoted as T_p in the rest of the paper.

Transactions arrivals are generally considered as a Poisson process as it is the common way to model them in literature [13][27]. We assume that the writes and the reads arrivals follow the Poisson distribution of parameter λ_w^{-1} (we chose λ_w^{-1} instead of λ_w in order to simplify subsequent formulas where the parameter will be inverted) and λ_r respectively. These parameters values change dynamically at run time following the read and write requests arrivals monitored in the storage system. Since the distribution of waiting time between two Poisson arrivals is an exponential process, the stochastic variables X_w and X_r of a write time and read time follow an exponential distribution of parameters λ_w^{-1} and λ_r respectively. The probability of the next read being stale corresponding to the aforementioned situation is given by formula (18) with N being the replication factor in the system and X being the number of replicas involved in the read operation. Here X_n =1 for the basic eventual consistency.

$$Pr(stale\_read) = \sum_{i=0}^{\infty}(\frac{N-(X_n=1)}{N} Pr(X_w^i < X_r < X_w^i + T + T_p) + \frac{X_n=1}{N} Pr(X_w^i < X_r < X_w^i + T)) \tag{18}$$

Having all the writes times (that may occur in the system) following the exponential distribution, the sum of X_w^i all the writes follows a $Gamma$ distribution of parameters i and λ_w. Hence, the probability in formula (18) becomes:

$$Pr(stale\_read) = \sum_{i=0}^{\infty}(\frac{N-1}{N} \int_0^{\infty} f_w^i(t)(F_r(t+T+T_p) - F_r(t))dt + \frac{1}{N} \int_0^{\infty} f_w^i(t)(F_r(t+T) - F_r(t))dt) \tag{19}$$

The time T to write in the local memory is negligible in comparison to T_P and therefore, we can consider it as equal to 0. A simple replacement of the probability mass function of Poisson distribution and the cumulative distribution function of $Gamma$ distribution results in the following probability:

$$Pr(stale\_read) = \sum_{i=0}^{\infty} \frac{N-1}{N} \int_0^{\infty} t^{i-1} \frac{e^{-\frac{t}{\lambda_w}}}{\gamma(i)\lambda_w^i}(e^{-\lambda_r t} - e^{-\lambda_r(t+T_p)})dt \tag{20}$$

After simplifying formula (20), it becomes:

$$Pr(stale\_read) = \sum_{i=0}^{\infty} \frac{(N-1)(1-e^{-\lambda_r T_P})}{N(1+\lambda_r\lambda_w)^i} \int_0^{\infty} t^{i-1} \frac{e^{\frac{1+\lambda_r\lambda_w}{\lambda_w}t}}{\gamma(i)(\frac{\lambda_w}{1+\lambda_r\lambda_w})^i}dt \tag{21}$$

The right part of the function in (21) is the the *cumulative distribution function* of a $Gamma$ law of parameters $\frac{1+\lambda_r\lambda_w}{\lambda_w}$ and i, its value is equal to 1. Moreover, if we consider that:

$$\sum_{i=0}^{\infty}(\frac{1}{1+\lambda_r\lambda_w})^i = \frac{1}{\lambda_r\lambda_w} + 1 \quad (22)$$

The final value of the probability of next read to be stale, after simplification, is given by:

$$Pr(stale\_read) = \frac{(N-1)(1-e^{-\lambda_r T_p})(1+\lambda_r\lambda_w)}{N\lambda_r\lambda_w} \quad (23)$$

Given that when the storage system supports multiple consistency levels, the consistency level for read and write operations (cl_r and cl_w respectively) may vary with time. Accordingly, we extend the probability model in Formula **??** to consider all the consistency levels for write operations that are smaller or equal to the *Quorum* level, where a quorum is computed as: $\lfloor \frac{replicationfactor}{2} + 1 \rfloor$. This probability is given in Formula 24.

$$Pr(stale\_read) = \frac{(N-(cl_w+cl_r-1))(1-e^{-\lambda_r T_p})(1+\lambda_r\lambda_w)}{N\lambda_r\lambda_w} \quad (24)$$

This estimation model requires basic knowledge of the application access pattern and of the storage system network latency. Network latency in this case is of high importance, since it is the determinant of the updates propagation time to other replicas. The access pattern, which includes read rates and write rates is a key factor to determine consistency requirements in the storage system. For instance, it is obvious that a heavy read-write access pattern would produce higher stale reads when adopting eventual consistency.

2) Efficiency-aware algorithm : Many applications do not strictly require strong consistency: a consistency optimization solution, therefore, can be introduced to improve system throughput, latency and monetary cost. To achieve this goal we consider our metric as an optimization metric as shown in the following algorithm.

while True **do**
  for $cl \in CLs$ **do**
    Compute $Cost_{rel}(cl)$
    Compute Consistency(cl)
    Compute Consistency(cl)/$Cost_{rel}(cl)$
  end for
  Choose $cl \in CLs$ for Max[Consistency(cl)/Cost(cl)]
end while

At run-time, our system feeds the efficiency-aware algorithm with data related to the system read/write rates along with the network latency. These data are used by the consistency probabilistic estimation model to compute the expected achieved fresh reads when using different consistency levels. The relative monetary cost is also computed according to the system configuration and the stale read estimation. So the algorithm selects the consistency level that offers the most equitable consistency, cost trade-off (the maximum consistency-cost efficiency value).

B. Implementation

We have built our approach as a separate layer on top of *Apache Cassandra-1.0.2* [16]. Cassandra gives the user flexible usage of consistency levels in a per-operation manner. In addition, Cassandra is proven to be very scalable, providing very good performance, and being widely used with large scale applications such as Facebook and Twitter. Our approach is introduced as an extra layer on Cassandra that aims to provide the most cost-efficient level of consistency for reading data. The core of this layer consists of two modules. Both modules were implemented in Python 2.7.
The monitoring module collects relevant metrics needed for our approach of the storage system's information. The Cassandra *nodetool* was used to collect the number of reads and writes in Cassandra storage, and we used the ping tool to collect network latencies in the storage system network (the network latency and the average data size is used to compute the propagation time T_p which later used in Formula 24). The monitoring module was designed in a multithreaded manner in order to make it time-efficient and to reduce the monitoring time. Each thread collects data from a set of nodes and at the end an aggregation process is applied. The monitoring time is measured and taken into account when computing the read rates and write rates. This data is further communicated to the **dynamic consistency module**. This module is the heart of our implementation. An estimation of cost-efficiency is computed — according to the estimated stale read rate and the monetary cost (instance, storage and network cost)— and then compared in order in to provide an adequate cost efficient consistency level (select the consistency level with the highest cost-efficiency ratio) for the running application at that point of time.

VI. Evaluation

In this section, we present our detailed evaluation of our the consistency-cost efficiency metric and the Bismar prototype using Cassandra on Grid'5000 testbed (We used the same testbed described in Section 3.2.). Our experiments evaluate two aspects: (a) validate the effectiveness of the consistency-cost efficiency metric as a representative metric for measuring; (b) overall monetary cost reduction achieved by Bismar (c) the tolerable fresh reads rates with Bismar and (d) the distribution of monetary cost amongst the different used resources when using Bismar.

Micro Benchmark. We use YCSB and we run WorkloadA which is a heavy read-update workload (read/update ratio: 60/40). Our workload consists of 10 million operations

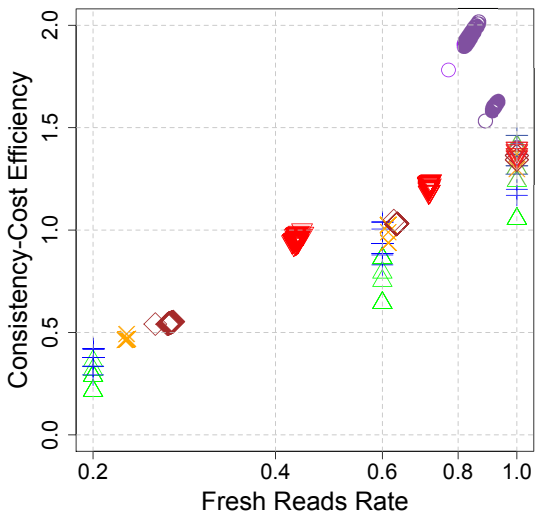


Figure 4. Effectiveness of the Consistency-cost efficiency

on 5 million rows with total of 23.84 GB of data after replication. In order to present the dynamicity of the system (i.e., the variation of throughput and the read/write rate during the run-time), we ran the workload, varying the number of threads starting with 1 thread, then, 50, 20, 7 and finally, 30 threads.

A. *Effectiveness of the Consistency-cost efficiency*

In order to validate our metric, we collect samples when running the same workload with different access patterns and different consistency levels. Figure 4 shows the results where each point shape represents a different access pattern: Higher consistency-cost efficiency values represents high rate of fresh reads for all samples when the fresh reads rate is lower or equal to 80%. This indicates the effectiveness of our metric: it is designed to achieve the best price without violating the consistency (we consider the 80% fresh reads as acceptable consistency).

B. *Monetary Cost and Performance improvement*

Figure 5(a) shows the monetary costs of running the workload with the three static consistencies (ONE, TWO and Quorum) and with our dynamic adaptive approach. As expected, ONE exhibits the lowest monetary cost but at the cost of fresh reads. Our experiments also show some interesting results: Bismar achieves lower cost in contrast to the consistency level TWO. Since *Bismar* always selects the consistency level with the highest consistency-cost efficiency to adapt to workload dynamicity, Bismar adopts the consistency level ONE for almost 70% of its running time while it adopts the consistency level Quorum for 30% of it is running time as shown in Figure 5(b). As a result the cost reduction when running with ONE overcomes the cost increase when running with Quorum.

Since *Bismar* targets applications that do not require strong consistency, we consider *Bismar* as an eventual consistency optimization for cloud. Therefore, improves the monetary cost of services while maintaining acceptable rate of fresh reads. Accordingly, we compare the cost monetary reduction and performance improvement by *Bismar* in contrast to the Quorum consistency level. As shown in Figure 5(a) and Figure 5(c), *Bismar* reduces the monetary cost by almost 31.5% in contrast to Quorum level (From $456 to $312). The cost reduction is mainly due to the performance improvements (*Bismar* improves the overall response time by almost 32.2%).However more detailed analysis on the impact of *Bismar* of the different resources' costs will be presented later.

C. *Staleness evaluation*

Figure 5(d) shows the stale reads rates caused by different consistency approaches. It is clear that static levels *ONE* and *TWO* produce higher stale reads rate: 61% of the reads where on stale data with ONE and 36% with TWO. Moreover, the Quorum consistency level returns always up-to-date data (i.e., stale reads rate is 0%) because at least one replica with the freshest data should be in the Quorum. *Bismar* however, returns very small portion of stale reads (only 3%), but with very important money saving (31.55% money reduction compared to Quorum). The 3% stale reads is considerably reasonable for many applications.

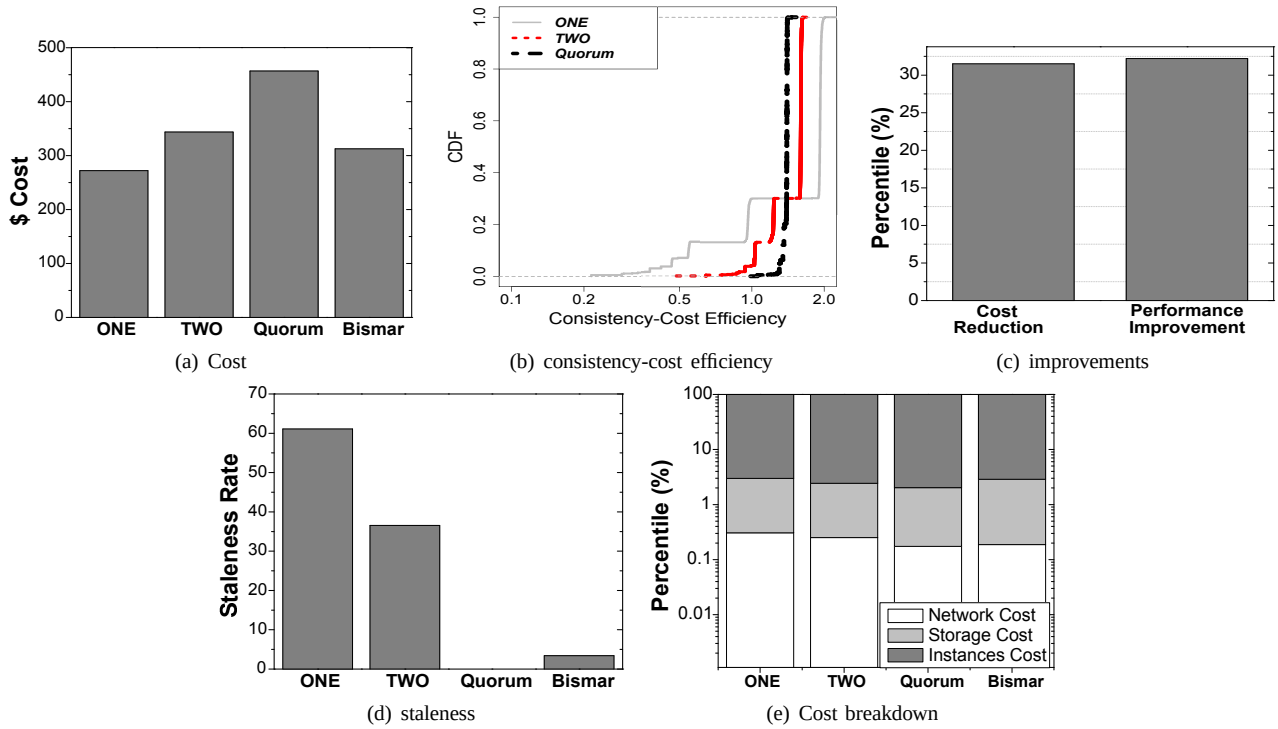
D. *Zoom on resources cost in* Bismar

Figure 5(e) shows the breakdown of the total cost according to the contributed resources for different consistency levels and *Bismar*. As shown and discussed earlier in Section 2.4, the instance portion of the total cost increases with upgrading consistency while the portion of both the storage and network costs increase with degrading the consistency level. However, the aforementioned observation is also applied on *Bismar*: comparing *Bismar* against Quorum, we notice that portion instance cost in *Bismar* is lower than in Quorum, furthermore, we observe that the portion of both the storage and network costs in *Bismar* is higher than in Quorum. This can explain why the cost reduction was only 31.5% while the performance improvement was 32.2%: because the adversary impacts of the storage and network costs in *Bismar*.

Moreover, we observe that the portion of both the storage and network costs in *Bismar* is higher than in all static consistencies, because *Bismar* combines both the high number of requests when adopting a higher consistency level and also read repair cost in the case of mismatched replicas' versions detection when *Bismar* adopts lower consistency level.

VII. Related Work

With the explosive growth of data size and availability requirements of services in the cloud along with the



(a) Cost (b) consistency-cost efficiency (c) improvements

(d) staleness (e) Cost breakdown

Figure 5. Cost, Staleness, and Efficiency on *Grid'5000*

tremendous increase in users accessing theses services, geographically distributed replication has become a necessity in the cloud storage [3][4][28]. At such scale, the strong consistency suffers of high latency and thus violating both the performance and availability requirements. Cloud storage is therefore evolving towards eventual consistency. Eventual consistency has been extensively exploited in literature and commercial products such as Dynamo [8] in Amazon S3 [29] and Amazon DynamoDB [30], Cassandra [16] in Facebook [12] and PNUTS [10] in Yahoo!. While the most of the work in literature have been dedicated to either measuring the actual provided consistency in cloud storage platforms [13][21][31], or on adaptive consistency tuning in cloud storage systems [32] [33] [34] [26] [6] in order to meet the consistency requirements of applications and reduce the consistency violation. Despite our work being focused on the monetary cost, a key difference between our work and their work is that we are seeking an adaptive consistency approach which is at the same time cost efficient and does not violate the applications needs.

A closely related work on improving the monetary cost of consistency in the cloud is [24]. Kraska et al. [24] propose consistency rationing: an automatic approach that adapts the level of consistency at run-time considering the performance and monetary cost. The authors define consistency levels at data level (i.e., categorizes the data into three types and provides a different consistency treatment for each category). Consistency rationing at data level may incur additional meta data management overhead when the data size is large, our work therefore is at a transaction level: our adaptive tuning approach chooses the number of replications involved in an operation considering the best trade-off between the consistency level and monetary cost. The results discussed in our paper complement Kraskas work: monetary cost-oriented consistency approach at transaction levels to complement their work at data level.

With respect to monetary cost in cloud systems, a number of studies [35][36] have been dedicated to measure the cost of adopting the pay-as-you-go cloud in terms of monetary cost, performance, and availability. Some studies [25][37][38] have reported on the cost variations and fairness in the cloud. Many recent studies concentrate on monetary cost improvements of cloud services through reducing the virtualization interference [39], using spot instance or levering the public cloud using free resources such as desktop grid [40][41]. In contrast, this paper investigates the interplay between economic issues and the consistency design and implementation.

VIII. Conclusion

With the pay-as-you-go charging, the public cloud has become an economic market for both cloud users and providers. Accordingly, many services have been deployed in the cloud in order to benefit from its low cost and its geographically distributed infrastructure: cloud allows

these services to replicate their data and thus satisfy the ever growing users' needs and ensure availability. However, ensuring data consistency between these geographically distributed replicas calls for empirical evaluations and technical innovations.

In this study, we investigate the monetary cost of consistency in the cloud. Our detailed analysis and study revealed a noticeable monetary cost variation when different consistency levels are used. As a first step to understand the impacts of the different consistency on the monetary cost and fresh reads in the cloud, we define the consistency-cost efficiency metric. Based on our metric, we introduce a simple, yet efficient approach, named *Bismar*, that adaptively tunes the consistency level at run-time in order to reduce the monetary cost while simultaneously maintaining a low fraction of stale reads. *Bismar* relies on a consistency probabilistic model that estimates the stale reads and the relative costs of the application according to the current read/write rate and network latency. We have implemented *Bismar* with intensive evaluations on Cassandra cloud storage system on Grid'5000. We show that *Bismar* can lead to efficient cost without exceeding the tolerated number of stale reads on the applications.

Regarding future work, we intend to perform more detailed theoretical and empirical analysis of the consistency-cost efficiency metric. Also we are interested in building an efficient mechanism for dynamic resource provisioning based on our cost function.

REFERENCES

[1] H. Jin, S. Ibrahim, T. Bell, L. Qi, H. Cao, S. Wu, and X. Shi, "Tools and technologies for building the clouds," *Cloud computing: Principles Systems and Applications*, pp. 3–20, Aug. 2010.

[2] "Amazon Elastic Compute Cloud (Amazon EC2)," November 2012. [Online]. Available: http://aws.amazon.com/ec2/

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *SIGOPS - Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th conference on usenix symposium on operating systems design and implementation*, 2006, pp. 205–218.

[5] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, "Maestro: Replica-aware map scheduling for mapreduce," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, Ottawa, Canada, 2012, pp. 59–72.

[6] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, "Making geo-replicated systems fast as possible, consistent when necessary," in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 265–278. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387906

[7] R. Peglar, "Eliminating planned downtime: the real impact and how to avoid it," May 2012. [Online]. Available: http://findarticles.com/p/articles/mi_m0BRZ/is_5_24/ai_n6095515/

[8] 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," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 205–220.

[9] "About Data Consistency in Cassandra," February 2012. [Online]. Available: http://www.datastax.com/docs/1.0/dml/data_consistency

[10] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. arno Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!s hosted data serving platform," In Proc. 34th VLDB, Tech. Rep., 2008.

[11] "Apache HBase," February 2012. [Online]. Available: http://hadoop.apache.org/hbase/

[12] "Facebook Statistcs," January 2012. [Online]. Available: http://newsroom.fb.com/content/default.aspx?NewsAreaId=22

[13] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu, "Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective," in *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, 2011, pp. 134–143.

[14] Y. Jégou, S. Lantéri, J. Leduc *et al.*, "Grid'5000: a large scale and highly reconfigurable experimental grid testbed." *Intl. Journal of High Performance Comp. Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[15] "Yahoo Cloud Serving Benchmark," February 2012. [Online]. Available: https://github.com/brianfrankcooper/YCSB/wiki

[16] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, April 2010.

[17] "Project Voldemort," May 2012. [Online]. Available: http://project-voldemort.com/

[18] "Riak: An Open Source Scalable Data Store," May 2012. [Online]. Available: http://wiki.basho.com/Riak.html

[19] M. Herlihy, "A quorum-consensus replication method for abstract data types," *ACM Trans. Comput. Syst.*, vol. 4, no. 1, pp. 32–53, Feb. 1986.

[20] D. K. Gifford, "Weighted voting for replicated data," in *Proceedings of the seventh ACM symposium on Operating systems principles*, ser. SOSP '79. New York, NY, USA: ACM, 1979, pp. 150–162.

[21] E. Anderson, X. Li, M. A. Shah, J. Tucek, and J. J. Wylie, "What consistency does your key-value store actually provide?" in *Proceedings of the Sixth international conference on Hot topics in system dependability*, ser. HotDep'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–16.

[22] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154.

[23] "mongoDB," February 2012. [Online]. Available: http://www.mongodb.org/

[24] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, "Consistency rationing in the cloud: pay only when it matters," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 253–264, Aug. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1687627.1687657

[25] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'10)*, Boston, MA, 2010.

[26] H.-E. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Pérez-Hernández, "Harmony: Towards automated self-adaptive consistency in cloud storage," in *2012 IEEE International Conference on Cluster Computing (CLUSTER'12)*, Beijing, China, 2012, pp. 293–301.

[27] A. T. Tai and J. F. Meyer, "Performability management in distributed database systems: An adaptive concurrency control protocol," in *Proceedings of the 4th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, ser. MASCOTS '96. Washington, DC, USA: IEEE Computer Society, 1996.

[28] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th USENIX conference on Symposium on Opearting Systems Design & Implementation (OSDI '04)*, San Francisco, CA, USA, 2004, pp. 137–150.

[29] "Amazon Simple Storage Service (Amazon S3)," February 2012. [Online]. Available: http://aws.amazon.com/s3/

[30] "Amazon DynamoDB," November 2012. [Online]. Available: http://aws.amazon.com/dynamodb/

[31] D. Bermbach and S. Tai, "Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior," in *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*, ser. MW4SOC '11. New York, NY, USA: ACM, 2011, pp. 1:1–1:6.

[32] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, "Consistency rationing in the cloud: pay only when it matters," *Proc. VLDB Endow.*, vol. 2, pp. 253–264, August 2009.

[33] S. Sakr, L. Zhao, H. Wada, and A. Liu, "Clouddb autoadmin: Towards a truly elastic cloud-based data store," in *Proceedings of the 2011 IEEE International Conference on Web Services*, ser. ICWS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 732–733.

[34] X. Wang, S. Yang, S. Wang, X. Niu, and J. Xu, "An application-based adaptive replica consistency for cloud storage," in *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, nov. 2010, pp. 13 –17.

[35] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon s3 for science grids: a viable solution?" in *Proceedings of the 2008 international workshop on Data-aware distributed computing*, ser. DADC '08. New York, NY, USA: ACM, 2008, pp. 55–64. [Online]. Available: http://doi.acm.org/10.1145/1383519.1383526

[36] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 50:1–50:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1413370.1413421

[37] S. L. Garfinkel, "An evaluation of amazons grid computing services: Ec2, s3 and sqs," Technical Report, Harvard University, Tech. Rep., 2007.

[38] S. Ibrahim, B. He, and H. Jin, "Towards pay-as-you-consume cloud computing," in *Proceedings of the 2011 IEEE International Conference on Services Computing (SCC'11)*, Washington, DC, USA, 2011, pp. 370–377.

[39] S. Ibrahim, H. Jin, L. Lu, B. He, and S. Wu, "Adaptive disk i/o scheduling for mapreduce in virtualized environment," in *Proceedings of the 2011 International Conference on Parallel Processing (ICPP'11)*, Taipei, Taiwan, 2011, pp. 335–344.

[40] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, Boston, MA, 2010.

[41] H. Liu, "Cutting mapreduce cost with spot market," in *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, Portland, OR, 2011, pp. 5–5.