Consistency in the Cloud:When Money Does Matter!

Houssem-Eddine Chihoub, Shadi Ibrahim, Gabriel Antoniu, María Pérez

Consistency in the Cloud: When Money Does Matter!

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

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

†Universidad Politécnica de 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 the 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 runtime in order to reduce the monetary cost while simultaneously maintaining a low fraction of stale reads. Experimental evaluations with the Cassandra cloud storage on the Grid'5000 testbed show the validity of the metric and demonstrate the effectiveness of the proposed consistency model.**

I. Introduction

Cloud computing has recently emerged as a popular paradigm for harnessing a large number of commodity machines. In this paradigm, users may rent 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. 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 environments in multiple physically separate datacenters to meet the ever-growing users' needs. Services therefore can replicate their state across geographically diverse sites and direct users to the closest or least loaded site. Replication has become an essential feature in storage systems and is extensively leveraged in cloud environments [1][2]. It stays behind several features such as fast access, enhanced performance, and high availability. For **fast access:** user requests can be directed to the closest datacenter in order to avoid communication delays and thus insure fast response time and low latency. For **enhanced performance:** in order to avoid overloading one single copy of the data, user requests can be re-directed to other replicas within the same datacenter (possibly different racks), to improve performance under heavy load. For **high availability:** failure and network partitions are common in large-scale distributed systems; by replicating, single points of failure can be avoided.

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 datacenters (the average round trip latency in Amazon sites varies from 0.3ms in the same site to 380ms in different sites [3]). 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 [4], Cassandra [5], BigTable [6], Yahoo! PNUTS [7], and HBase [8]. These solutions are practical to use as cloud and web service storage backends. They allow many web services to scale out their systems in an extreme way, while maintaining performance with very high availability. For example, Facebook uses Cassandra to scale out to host data for more than 800 million active users [9]. However, the undoubted availability and performance of such solutions prove to be too costly in terms of inconsistency ([10] reports the stale read rate was 66.61% under heavy access). Consistency-performance and consistency-availability trade-offs have long been investigated in literature: many consistency optimization solutions have been devoted to improving the application throughput and/or latency while preserving acceptable stale reads rate.

However, in the area of cloud computing, the economic 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: **[i] 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 service into three parts with respect to the contributed resources: virtual machine (VM) instances cost, storage cost and network cost. To complement our analysis, a series of experiments are conducted to measure the monetary cost of different consistency levels in the Cassandra system [5] on Grid'5000 [11] and Amazon EC2 [12]. 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. **[ii] Novel metric.** We define a new metric called consistency-cost efficiency to evaluate consistency in the cloud. **[iii] 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 runtime 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 rate 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 the Cassandra cloud storage system on Grid'5000 [11]. We use the Yahoo! Cloud Serving Benchmark (YCSB) [13] to mimic a real cloud serving environment with elastic access pattern workloads. We show that *Bismar* can lead to efficient costs without exceeding the number of stale reads tolerated by the applications. Our paper is the first to provide a thorough analysis of the consistency cost in cloud storage systems. We view our work 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 more economical consistency models for cloud services.

II. Related Work and motivation

A. Related Work

Eventual consistency has been extensively exploited in literature and commercial products such as Dynamo [4] in Amazon, Cassandra [5] in Facebook and PNUTS [7] in Yahoo!. Most of the work in literature have been dedicated to either measuring the actual provided consistency in cloud storage platforms [10][14][15], or on adaptive consistency tuning in cloud storage systems [16][17] in order to meet the performance requirements of applications and reduce the consistency violation. In contrast, we focus on the monetary cost: we aim to provide an adaptive consistency approach that is cost efficient and does not violate the applications needs.

A closely related work on improving the monetary cost of consistency in the cloud is [18]. *Kraska et al.* [18] propose consistency rationing: an approach that adapts the level of consistency at runtime considering the 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 metadata management overhead when the data size is large, our work therefore is at a transaction level: our adaptive tuning approach selects the number of replicas involved in an operation considering the best trade-off between consistency and monetary cost. The results discussed in our paper complement Kraska' work: monetary cost-oriented consistency approach at transaction levels to complement their work at data level.

B. Monetary Cost of Consistency: Why it does matter!

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 operation latency causes high monetary cost. Obviously because the cost of leasing a VM-instance is proportional to the latency (runtime). In addition to the increased cost of both the storage (e.g., number of IO requests) and the communication cost (e.g., number of cross-sites communication) due to the synchronous cross-site replication, (2) High operation 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$-3.1M$ [14]. 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., [10] demonstrated that under heavy access some of these systems may return up to 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 general, services require a set of linked servers (distributed in multi-sites) to run the applications; these servers are attached to a group of storage devices which store services data. With respect to cloud resources offers, a basic service bill will include charges for the following resources: **[i] Computing resources.** Virtual machines equipped with a certain amount of CPU and memory resources and typically charged for the incurred virtual machine hours. **[ii] Storage resources.** 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. **[iii] Network resources.** 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.

Hereafter, we present a detailed analysis of the consistency cost in the cloud, using a widely used open source cloud storage system that supports multi-level consistency as an illustrated example, namely Cassandra [5]. In Cassandra, the consistency level may be chosen on a per-operation basis and is represented by the number of replicas in the quorum (a subset of all the replicas). 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 a 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 services with geo-distributed replication for a given consistency level cl. Essentially, this cost is the combination of the VM instances cost Cost$_{in}$(cl), the backend storage cost Cost$_{st}$(cl), and network cost Cost$_{tr}$ (cl).

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

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).

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

Here the price is the dollar cost per timeUnit[1] (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 , we present the runtime in the form of number of operations $nbOps$ in the workload while fixing the average throughput(ops/s) of a specific consistency level (runtime $= \frac{\text{nbOps}}{\text{throughput}}$). 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 (GBs 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 3.

$$\text{Cost}_{st}(\text{cl}) = \text{costPhysicalHosting} + \text{costIORequests} \qquad (3)$$

Based on the size of hosted data (including all data replicas) 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 datacenters), we calculate the costPhysicalHosting in Formula 4.

$$\text{costPhysicalHosting} = \text{nbNodes} \times \lceil\frac{\text{dataSize}}{\text{sizeUnit}}\rceil \times \text{price} \qquad (4)$$

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 5.

$$\text{costIORequests} = \frac{\text{cl} \times \text{nbOps} + \text{readRepairIO}}{\text{nbRequestsUnit}} \times \text{price} \qquad (5)$$

where nbOps is the number of operations with respect to the consistency level cl (it varies according to the number of replicas involved in an operation). readRepairIO is the number of operations generated by the *read repair* operations. The read repair is a background operation that is mostly triggered when inconsistency is detected (more details about the read repair will be provided in Section III-C).

C. Network cost

Network cost varies in accordance to the service type of the source and destination and whether the data transfer is within or across sites. In general, inter-datacenter communications are more expensive than intra-datacenter communications. Formula 6 shows the total cost of network communications[2] as the sum of inter-datacenter (trafficInterDC

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

[2] 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.

and intra-datacenter trafficIntraDC communications cost.

$$\begin{aligned}\text{Cost}_{tr}\ (\text{cl}) = \text{price(interDC)} \times \lceil \frac{\text{trafficInterDC}}{\text{sizeUnit}} \rceil \\ + \text{price(intraDC)} \times \lceil \frac{\text{trafficIntraDC}}{\text{sizeUnit}} \rceil\end{aligned} \tag{6}$$

Hereafter, we illustrate how to estimate both the inter- and intra-datacenter traffic. Formula 7 shows our model of the inter datacenter, trafficInterDC, given the replicas communication (interDcRep), the request routing (request routing), and the internal mechanisms traffic (IMechTraffic).

$$\text{trafficInterDC} = \text{interDcRep} + \text{requestRouting} + \text{IMechTraffic} \tag{7}$$

The inter-site traffic generated by the replicas communications strongly depends on the cl and the distribution of replicas among datacenters (i.e., the number of replicas involved in a request to other datacenters which can be estimated as $\lfloor(\text{nbDc - 1})\times\frac{\text{cl}}{\text{nbDc}}\rfloor$[3] where nbDc is the number of datacenters). Formula 8 shows our estimation of the inter traffic generated by the replicas communications.

$$\text{InterDcRep} = \lfloor(\text{nbDc - 1}) \times \frac{\text{cl}}{\text{nbDc}}\rfloor \times \text{AvgDataSize} \times \text{nbOps} \tag{8}$$

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.

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 9.

$$\begin{aligned}\text{requestRouting(interDC)} = \frac{\text{nbNodes} - \text{avgNodesDc}}{\text{nbNodes}} \\ \times\text{nbOps} \times \text{avgDataSize}\end{aligned} \tag{9}$$

In Cassandra, the main internal traffic is generated by the gossip traffic and the read repair as shown in Formula 10.

$$\text{IMechTraffic} = \text{gossip(interDc)} + \text{readRepair(interDc)} \tag{10}$$

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. On the other hand, the read repair is used to propagate data to out-of-date (stale) replicas. The read repair is triggered in two cases: (1) At random times for some requests: defined by the system administrator; (2) Whenever inconsistency is detected. Formula 11 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 writes and reads.

$$\begin{aligned}\text{readRepair(interDC)} = \text{nbOps} \times \text{avgDataSize} \\ \times(\text{rrChance} \times \lfloor\frac{\text{rf}}{\text{nbDc}}\rfloor + \text{mmChance} \times \lfloor\frac{\text{rf} - \text{cl}}{\text{nbDc}}\rfloor)\end{aligned} \tag{11}$$

Computing the intra datacenter traffic size is very similar to the one of inter datacenter traffic. Due to the page limitation we didn't include its analysis here (Please refer to our technical report [19] for more details).

D. Practical View of Consistency Cost in Cassandra

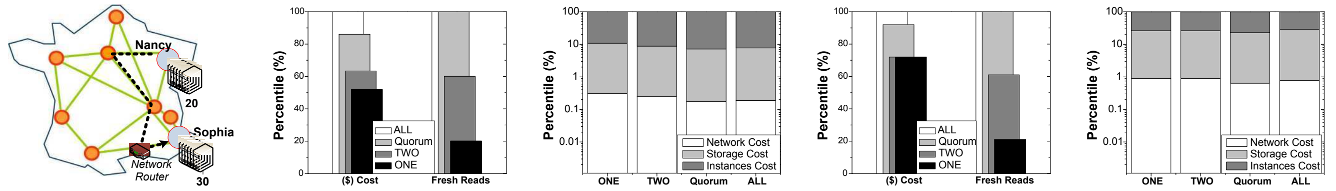
We complement and benefit from our earlier analyze, by evaluating the monetary cost in Cassandra.

Experimental setup. We run our experiments on Grid'5000 [11] and Amazon Elastic Compute Cloud (EC2) [12]. 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 1(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[4]. We study the cost variation by evaluating different consistency levels (e.g., eventual consistency: ONE, TWO, Quorum: THREE, and strong consistency: ALL).

[3] 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.

[4] 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.



(a) Experiments setup on Grid5000

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

(c) Breakdown of the Monetary cost on Grid'5000 (*The y-axis is logarithmic*)

(d) Monetary Cost and Fresh reads rate on Amazon EC2

(e) Breakdown of the Monetary cost on Amazon EC2 (*The y-axis is logarithmic*)

Figure 1. Experiments setup and results on *Grid'5000* and Amazon EC2

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

Micro Benchmark. We aim at a micro benchmark representing typical workloads in current services hosted in clouds. Based on case studies [7][20], we have selected the Yahoo! Cloud Serving Benchmark (YCSB) [13]. YCSB is used to benchmark Yahoo! storage system "PNUTS" [7]. It is extended to be used with a variety of *open-source* data stores such as mongoDB [21] and Cassandra [5]. YCSB provides the features of a real cloud serving environment such as scale-out, elasticity and high availability. 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.

Results. As shown in Figure 1(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 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 runtime of WorkloadA varies from 4 hours to 7 hours 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 1(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 to 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 1(b). This is because the storage system intends to answer the read requests with the most up-to-date replicas (fresh reads). 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. Figure 1(c) shows the breakdown of the total cost according 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 1(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. *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 datacenter if possible") while both have zero/low IMechTraffic and thus according to Formula 10 ALL has a relatively higher network cost contribution in contrast to Quorum. Moreover, although the interDcRep is smaller 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.

Amazon EC2: Figures 1(d) and 1(e) support our earlier findings and observations with Grid'5000 (for more details please refer to [19]). The total cost variation in Amazon is lower than in Grid'5000, because of the more powerful machines and the lower cross-sites latency. Moreover, the costs of the ONE and TWO levels are the same, although there were significant variations in the running time (2hours and 1minute 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 (hour and GBs , etc).

IV. Consistency-Cost Efficiency Metric

As discussed in the previous sections, data consistency can strongly impact the financial cost of a given service (i.e., while stronger consistency with high latency implies higher monetary cost of an operation, the weaker consistency with high rate of stale rate causes financial loss). Consequently, monetary cost should be considered when evaluating the consistency in the cloud [18]. As cloud computing is an economy-driven distributed system where monetary cost is explicit and measurable metric [22][23], 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 a 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 12.

$$\text{Consistency-Cost Efficiency} = \frac{\text{Consistency}(cl)}{\text{Cost}_{rel}(cl)} \tag{12}$$

Where $Consistency(cl) = 1 - stale\ reads\ rate$ and Cost_{rel} is the relative consistency cost with respect to the strong consistency and given as $\text{Cost}_{rel}(\text{c}l) = \frac{\text{Cost}(cl)}{\text{Cost}(cl\_all)}$.

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 a 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. *BISMAR*: Economical Consistency Approach

We design and implement our approach with the following goals: **[A]** *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. **[B]** *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 [17] 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, we use the same information (e.g., access pattern and network latency) along with the predicted stale read rate (i.e., to estimate the number of stale reads) to compute the monetary cost. **[C]** *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. **[D]** *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.

Consistency Probabilistic Estimation. In our previous work [17], we proposed an estimation of the stale reads rate in the system by means of probabilistic computations. 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.

Hereafter, we briefly describe our extended estimation model (for more details readers can refer to [17]). Essentially, the client gets stale data when a read operation is performed from one or more replicas that have not been updated yet, while an update process is going on. Transactions arrivals are almost exclusively considered as *Poisson* process as it is a common way to model them [17]: 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. Since the distribution of waiting time between two *Poisson* arrivals is an exponential process. We assume the stochastic variables X_w and X_r of write time date and read time date follow exponential distributions of parameters λ_w^{-1} and λ_r respectively.

The probability of a stale read $Pr(staleRd)$, assuming that writes are performed with a consistency level that involves only one replica, is given by Formula 13 where rf is the replication factor, and T_p is the average time to propagate an update to other replicas.

$$Pr(staleRd) = \frac{(rf-1)(1-e^{-\lambda_r T_p})(1+\lambda_r\lambda_w)}{rf\lambda_r\lambda_w} \qquad (13)$$

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 13 to consider all the consistency levels for write and read 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 14.

$$Pr(staleRd) = \frac{(rf-(cl_w+cl_r-1))(1-e^{-\lambda_r T_p})(1+\lambda_r\lambda_w)}{rf\lambda_r\lambda_w} \qquad (14)$$

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 runtime, 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).

VI. Evaluation

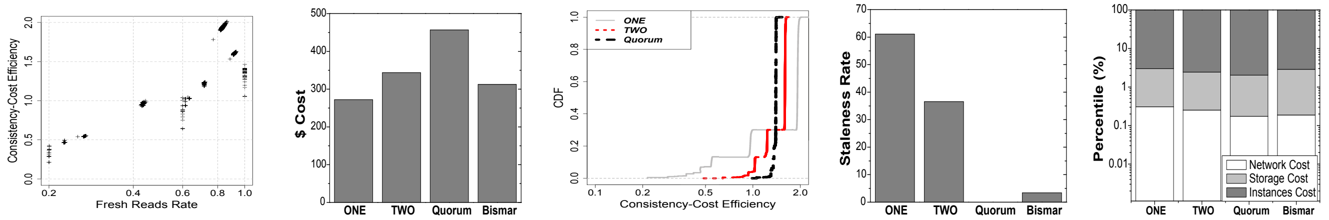
We have built our approach as a separate layer at the top of *Apache Cassandra-1.0.2* [5]. The core of this layer consists of two modules. Both modules were implemented in Python 2.7. *The monitoring module* collects relevant metrics (data) needed for our approach of the storage system's information. The data is further communicated to the *dynamic consistency module*. An estimation of consistency-cost efficiency is computed — according to the estimated stale reads rate and the monetary cost (instance, storage and network cost)— and then compared in order to provide a cost efficient consistency level for the running application at that point of time. Later in this section, we present our detailed evaluation of our consistency-cost efficiency metric and the *Bismar* prototype using Cassandra on Grid'5000 testbed. We also use YCSB to run WorkloadA (We used the same testbed and WorkloadA described in Section III-D). In order to present the dynamicity of the system (i.e., the variation of throughput and the read/write rates during the runtime), we ran the workload, varying the number of threads starting with 1 thread, then, 50, 20, 7 and finally, 30 threads.

Effectiveness of the Consistency-cost efficiency. In order to validate our metric, we collect samples when running the same workload (with different consistency levels), varying the number of client threads, and thus exhibiting different access patterns. Figure 2(a) shows the results where each point represents a different access pattern. Higher consistency-cost efficiency values are associated with high rates of fresh reads (around 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).

Monetary Cost and Performance improvement. Figure 2(b) 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 adopt to the 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 its running time as shown in Figure 2(c). 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 computing. Therefore, improves the monetary cost of services while maintaining acceptable rate of fresh reads. Accordingly, we compare the cost reduction and performance improvement by *Bismar* in contrast to the Quorum consistency level. As shown in Figure 2(b), *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%).

Staleness evaluation. Figure 2(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% cost reduction compared to Quorum). The 3% stale reads is considerably reasonable for many applications.

Zoom on resources cost in *Bismar*. Figure 2(e) shows the breakdown of the total cost according to the contributed



(a) CCE-Effectiveness

(b) *Bismar :* Monetary Cost

(c) CCE-Distribution

(d) *Bismar :* Staleness

(e) *Bismar :* Cost breakdown (*The y-axis is logarithmic*)

Figure 2. (a) and (c) consistency-cost efficiency CCE-effectiveness and -distribution; (b), (d), and (e) *Bismar* evaluation on *Gird'5000*

resources for different consistency levels and *Bismar*. As shown and discussed earlier in Section III-D, 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 instance cost portion 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 of 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 when stale reads is detected when *Bismar* adopts lower consistency level.

VII. Conclusion

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 consistencies 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 extensive evaluations on the Cassandra cloud storage system using the Grid'5000 testbed. We show that *Bismar* can lead to efficient cost without exceeding the application tolerated number of stale reads. As future work, we intend to perform more detailed theoretical and empirical analysis of the consistency-cost efficiency metric with a broader set of application workloads. Also we are interested in building an efficient mechanism for dynamic resource provisioning based on our cost function.

Acknowledgments

This work is supported by the EU FP7 MCITN SCALUS project under grant agreement no. 238808, the Héméra INRIA Large Wingspan-Project, and the ANR MapReduce grant (ANR-10-SEGI-001). The experiments presented in this paper were carried out using the Grid'5000/ALADDIN-G5K experimental testbed.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *SIGOPS - Operating Systems Review*, pp. 29–43, 2003.

[2] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, "Maestro: Replica-aware map scheduling for mapreduce," in *CCGrid '12*, Ottawa, Canada, 2012, pp. 59–72.

[3] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, "Making geo-replicated systems fast as possible, consistent when necessary," in *OSDI'12*.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *SOSP '07*, pp. 205–220.

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

[6] 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 *SOSP'06*, pp. 205–218.

[7] 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.

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

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

[10] 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*, pp. 134–143.

[11] 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*, pp. 481–494, 2006.

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

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

[14] E. Anderson, X. Li, M. A. Shah, J. Tucek, and J. J. Wylie, "What consistency does your key-value store actually provide?" in *HotDep'10*, pp. 1–16.

[15] D. Bermbach and S. Tai, "Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior," in *MW4SOC '11*, pp. 1:1–1:6.

[16] S. Sakr, L. Zhao, H. Wada, and A. Liu, "Clouddb autoadmin: Towards a truly elastic cloud-based data store," in *ICWS '11*, pp. 732–733.

[17] H.-E. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Pérez-Hernández, "Harmony: Towards automated self-adaptive consistency in cloud storage," in *CLUSTER'12*.

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

[19] H.-E. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Pérez-Hernández, "Consistency in the cloud: When money does matter!" in *Technical Report*. [Online]. Available: http://hal.inria.fr/hal-00756314/

[20] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *SoCC '10*, pp. 143–154.

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

[22] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *HotCloud'10*.

[23] S. Ibrahim, B. He, and H. Jin, "Towards pay-as-you-consume cloud computing," in *SCC'11*, pp. 370–377.