Interplaying Cassandra NoSQL Consistency and Performance: a Benchmarking Approach

Anatoliy Gorbenko
Leeds Beckett University
Leeds, UK
A.Gorbenko@leedsbeckett.ac.uk

Alexander Romanovsky
Newcastle University
Newcastle-upon-Tyne, UK
Alexander.Romanovsky@ncl.ac.uk

Olga Tarasyuk
National Aerospace University
Kharkiv, Ukraine
O.Tarasyuk@csn.khai.edu

***Abstract*—This experience report analyses performance of the Cassandra NoSQL database and studies the fundamental trade-off between data consistency and delays in distributed data storages. The primary focus is on investigating the interplay between the Cassandra performance (response time) and its consistency settings. The paper reports the results of the read and write performance benchmarking for a replicated Cassandra cluster, deployed in the Amazon EC2 Cloud. We present quantitative results showing how different consistency settings affect the Cassandra performance under different workloads. One of our main findings is that it is possible to minimize Cassandra delays and still guarantee the strong data consistency by optimal coordination of consistency settings for both read and write requests. Our experiments show that (i) strong consistency costs up to 25% of performance and (ii) the best setting for strong consistency depends on the ratio of read and write operations. Finally, we generalize our experience by proposing a benchmarking-based methodology for run-time optimization of consistency settings to achieve the maximum Cassandra performance and still guarantee the strong data consistency under mixed workloads.**

Keywords—NoSQL, Cassandra database, CAP theorem, trade-off, consistency, performance, latency, benchmarking

I. INTRODUCTION

NoSQL (Non SQL or Not Only SQL) databases have become the standard data plat-form and a major industrial technology for dealing with enormous data growth. They are now widely used in different market niches, including social networks and other large-scale Internet applications, critical infrastructures, business-critical systems, IoT and industrial applications. NoSQL databases designed to provide horizontal scalability are often offered as a service by Cloud providers. The concept of NoSQL databases [1] has been proposed to effectively store and provide fast access to the Big Data sets whose volume, velocity and variability are difficult to deal with by using the traditional Relational Database Management Systems. Most NoSQL stores sacrifice the ACID (atomicity, consistency, isolation and durability) guarantees in favour of the BASE (basically available, soft state, eventually consistent) properties [2], which is the price to pay for distributed data handling and horizontal scalability.

The paper discusses the trade-offs between consistency, availability and latency, which is in the very nature of NoSQL databases. Although these relations have been identified by the CAP theorem in qualitative terms [3, 4], it is still necessary to quantify how different consistency settings affect system latency and throughput. Understanding this trade-off is key for the effective usage of NoSQL solutions. While there are many NoSQL databases on the market, various industry trends suggest that Apache Cassandra is one of the top three in use today together with MongoDB and HBase [5]. There have been a number of studies, e.g. [6, 7, 8, 9, 10], evaluating and comparing the performance of different NoSQL databases. Most of them use general competitive benchmarks of usual-and-customary application workloads (e.g. Yahoo! Cloud Serving Benchmark, YCSB). The reported results show that depending on the use case scenario, deployment conditions, current workload and database settings any NoSQL database can outperform the others. Other recent related works, such as [11, 12, 13], investigate the measurement-based performance prediction of NoSQL data stores. However, the studies mentioned above, do not analyse an interdependency between consistency and performance, that is in the very nature of such distributed database systems and do not study how consistency settings affect database latency.

In this paper we put a special focus on quantitative evaluation of the fundamental Big Data trade-offs between data consistency and performance using the Cassandra database as a typical example of distributed data storages. Apache Cassandra offers a set of unique features (e.g. tuneable consistency, extremely fast writes, ability to work across geographically distributed data centres, etc.) and provides high availability with no single point of failure which makes it one of the most flexible and popular NoSQL solutions. Moreover, we would like to equip the developers of distributed systems that use Cassandra as the distributed data storage with the practical guidance allowing them to predict the Cassandra latency taking into account the required consistency level and to coordinate consistency settings of read and write requests in an optimal manner. In the paper we propose a benchmarking approach to optimising the Cassandra performance with guaranteeing the strong data consistency under mixed workloads. Because the real workload mix can evolve and change over time impacting the desirable Cassandra settings, we propose to monitor the current workload mix and chose the optimal consistency setting at run time to get the highest throughput by making use of benchmarking results collected during system load testing.

The rest of the paper is organized as follows. In the next section, we briefly discuss the fundamental CAP trade-off for distributed systems and replicated data storages, and analyse the Cassandra tuneable consistency feature. In Section III, we describe our methodology and the experimental setup, and present the results of the Cassandra performance benchmarking for different consistency levels. Sections IV and V discuss the optimal consistency settings and propose a methodology for optimal coordination of consistency settings for read and write requests under different workloads. Finally, some practical lessons learnt from our work are summarized in Section VI.

II. BIG DATA TRADE-OFFS BETWEEN CONSISTENCY, AVAILABILITY AND LATENCY

A. CAP Theorem

The CAP conjecture [3], which first appeared in 1998-1999, defines a trade-off between system availability, consistency and partition tolerance, stating that only two of the

three properties can be preserved in distributed replicated systems at the same time. Gilbert and Lynch [4] view the CAP theorem as a particular case of a more general trade-off between *consistency*, *availability* and *latency* in unreliable distributed systems which, nevertheless, assume that updates are eventually propagated.

System partitioning, availability, consistency and latency (response time) are tightly connected. Moreover, we believe that these properties need to be viewed as more continuous than binary. A replicated fault-tolerant system becomes *partitioned* when one of its parts does not respond due to arbitrary message loss, delay or replica failure, resulting in a timeout. System *availability* can be interpreted as the probability that each client request eventually receives a response. Failure to receive responses from some of the replicas within the specified timeout causes partitioning of the replicated system. Thus, partitioning can be considered as a bound on the replica *latency*/*response time* [14, 15]. A slow network connection, a slow-responding replica or the wrong timeout settings can lead to an erroneous decision that the system has become partitioned. When the system detects a partition, it has to decide whether to return a possibly inconsistent response to a client or to send an exception message in reply, which undermines system availability. *Consistency* is also a continuum, ranging from weak consistency at one extreme to strong consistency on the other, with varying points of eventual consistency in between.

The designers of distributed fault-tolerant systems cannot prevent partitions which happen due to network failures, message losses, hacker attacks or components crashes and, hence, have to choose between availability and consistency. One of these two properties has to be sacrificed. The architects of modern distributed database management systems and large-scale web applications such as Facebook, Twitter, etc. often decide to relax consistency requirements by introducing asynchronous data updates in order to achieve higher system availability and allow a quick response. Yet the most promising approach is to balance these properties [16, 17]. For instance, the Cassandra NoSQL database supports a tuneable replication factor and an adjustable consistency model so that a user can choose a particular level of consistency to fit with the desired system latency.

B. Cassandra's Tuneable Consistency

The Cassandra NoSQL database extends the concepts of *strong* [18] and *eventual* [19] consistency by offering *tuneable* [20] consistency. Consistency in Cassandra can be configured to trade-off availability and latency versus data consistency.

The consistency level among replicated nodes can be controlled on a per-operation basis. Thus, for any given read or write operation, a client can specify how consistent the requested data must be. The *read consistency level* specifies how many replica nodes must respond to a read request before returning data to the client application. In turn, the *write consistency level* determines the number of replicas on which the write must succeed before returning an acknowledgment to the client.

It is worth noting that Cassandra supports two types of write operations with the tiny difference between them: insert and update. Cassandra treats both insert or update operations as *upserts* (update-or-insert) [21]. It adds each new row to the database without really checking on whether a duplicate record exists. This makes it possible that many versions of the same row may exist in the database. Periodically, the rows stored in memory (in a structure called *memtable*) are streamed to disk into structures called *SSTables*. At certain intervals, Cassandra compacts smaller *SSTables* into larger *SSTables*. If Cassandra encounters two or more versions of the same row during this process, it only writes the most recent version to the new *SSTable* and drops the original *SSTables*, deleting the outdated rows.

All Cassandra read and write requests support the following basic consistency settings [22]:

- ONE: data must be written to the commit log and *memtable* of at least one replica node before acknowledging the write operations to a client; when reading data, Cassandra queries and returns a response from a single replica (the nearest replica with the least network latency);
- TWO: data must be written to at least two replica nodes before being acknowledged; read operations will return the most recent record from two of the closest replicas (the most recent data is determined by comparing timestamps of records returned by those two replica);
- THREE: similar to TWO but for three replicas;
- QUORUM: a quorum of nodes needs to acknowledge the write or to return a response for a read request; a quorum is calculated by rounding down to a whole number the following estimate: *replication_factor*/2+1;
- ALL: data must be written to all replica nodes in a cluster before being acknowledged; read requests return the most recent record after all replicas have responded. The read operation will fail even if a single replica does not respond.

If Cassandra runs across multiple data centres, a few additional consistency levels become available: EACH_QUORUM, LOCAL_QUORUM, LOCAL_ONE.

The sum of nodes written and read being greater than the replication factor always ensures strong data consistency when a read never misses a preceding write [22]. Thus, if data consistency is of a top priority, one can ensure that a read always reflects the most recent updates by using the following:

$$(nodes\_written + nodes\_read) > replication\_factor \quad (1)$$

otherwise, the eventual consistency occurs. For example, if Cassandra uses a replication factor of 3, the strong consistency is ensured if, either:

- the QUORUM consistency level is set for both write and read requests;
- the ONE consistency level is set for writes and ALL for reads or;
- the ALL consistency level is set for writes and ONE for reads.

The weaker consistency level, the faster Cassandra should perform read and write requests. Balancing between *nodes_written* and *nodes_read* in (1), Cassandra users can give the priority to read or write performance still guaranteeing the strong data consistence.

III. CASSANDRA PERFORMANCE BENCHMARKING

A. Methodology and Experimental Setup

In this section we describe our performance benchmarking methodology and report the experimental results showing how consistency settings affect latency of the read and write requests for the Cassandra NoSQL database.

Cassandra deployment setup. As a testbed we deploy the 3-replicated Cassandra 2.1 cluster in the Amazon EC2 cloud (Fig. 2). Replication factor equal to 3 is the most typical setup for many modern distributed computing systems and Internet services, including Amazon S3, Amazon EMR, Facebook Haystack, DynamoDB, etc. The cluster is deployed in the AWS US-West-2 (Oregon) region on c3.xlarge instances (vCPUs – 4, RAM – 7.5 GB, SSD – 2x40 GB, OS – Ubuntu Server 16.04 LTS).

Benchmark. Our work uses the YCSB (Yahoo! Cloud Serving Benchmark) framework which is considered to be a de-facto standard benchmark to evaluate performance of various NoSQL databases like Cassandra, MongoDB, Redis, HBase and others [6]. YCSB is an open-source Java project.

The YCSB framework includes six out-of-the-box workloads [6], each testing different common use case scenarios with a certain mix of reads and writes (50/50, 95/5, read-only, read-latest, read-modify-write, etc.). In our experiments we use the read-only Workload C, and the Workload A, which is parametrized to execute write-only operations.

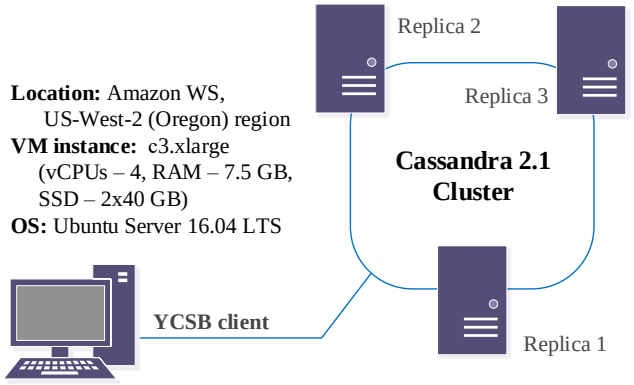

Fig. 1. Experimental setup: Cassandra cluster.

All the rest Cassandra and YCSB parameters (e.g. request distribution, testbed database, etc.) were set to their default values. The testbed YCSB database is a table of records. Each record is identified by a primary key and includes F string fields. The values written to these fields are random ASCII strings of length L. By default, F is equal 10 and L is equal 100, which constructs 1000 bytes records. The final size of the testbed database reached 70GB by the end of our experiments.

The YCSB Client is a Java program that generates data to be loaded to the database, and runs the workloads. It is deployed on a separate VM in the same Amazon region to reduce the influence of the unstable Internet delays.

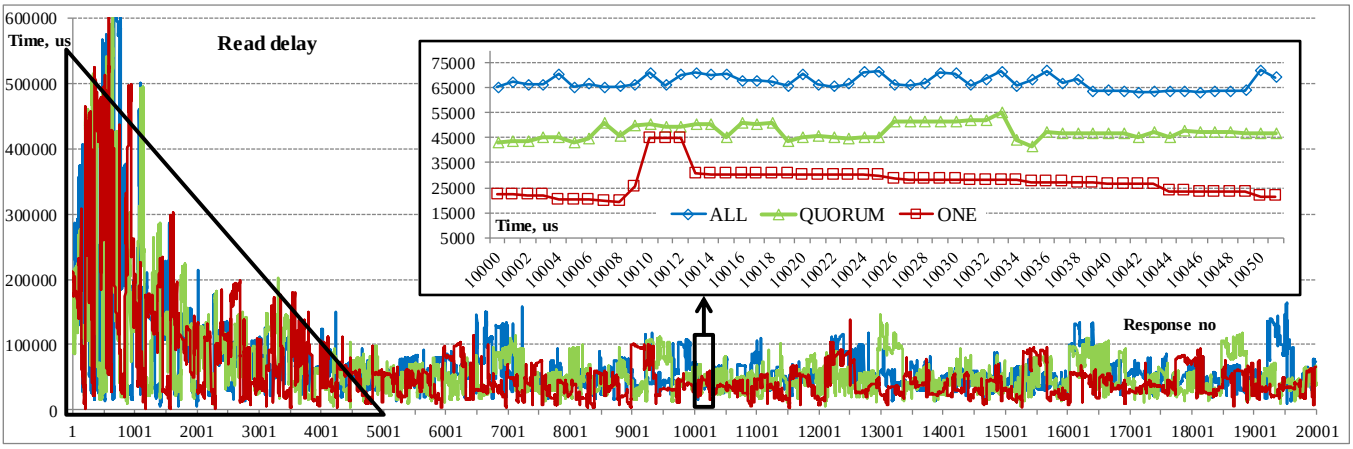

Fig. 2. A fragment of the READ delay graph, 500 threads.

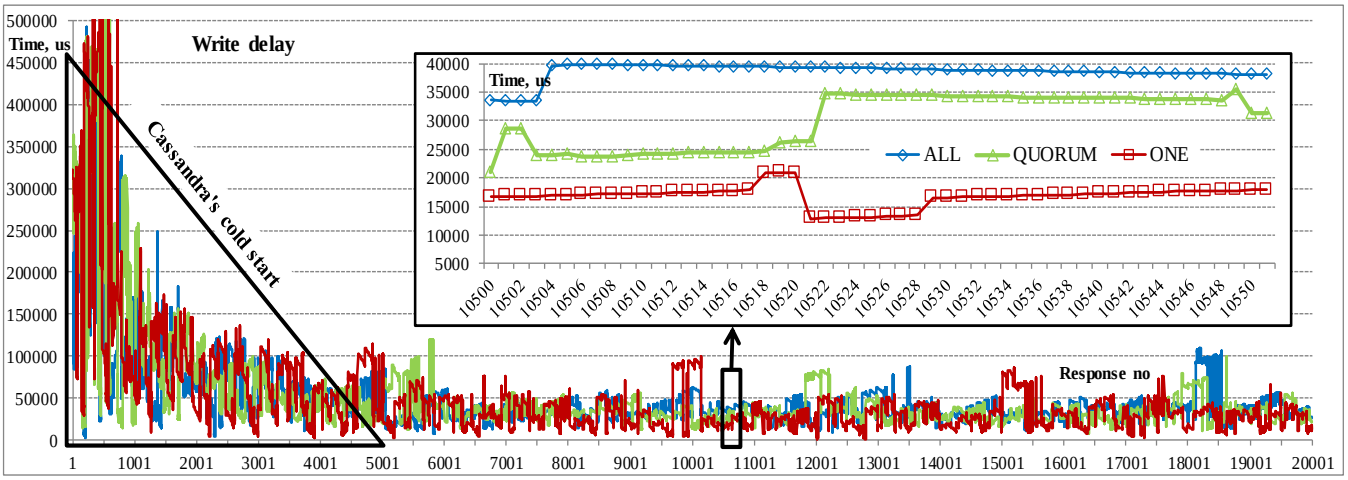

Fig. 3. A fragment of the WRITE delay graph, 500 threads.

Benchmarking scenario. Some examples of general methodologies for benchmarking Cassandra and other NoSQL databases with YCSB can be found in [5, 23]. However, unlike these and other works (e.g. [6, 7, 8, 9, 10]) studying and comparing the maximum databases throughput we put the focus on analysing the dynamic aspects of the Cassandra performance under different consistency settings. In particular, we analyse how the database latency and throughput depend on a current workload (i.e. number of concurrent requests/threads). To achieve this we run a series of YSCB read and write performance tests on Apache Cassandra with a number of threads varying from 10 to 1000. The operation count within each thread is set to 1000. The same scenario is run for read and write workloads on the 3-replicated Cassandra cluster with the three different consistency settings: ONE, QUORUM and ALL.

B. Raw Data Cleansiing

YCSB supports different measurement types including 'histogram', 'timeseries' and 'raw'. In our experiments we set it to 'raw' when all measurements are output as raw data points in the following *csv* format: operation (READ|WRITE), timestamp of the measurement (ms), latency (us). This allows us to plot the response delay graphs, see the examples in Figs. 2 and 3 (each graph superimposes three curves corresponding to different consistency settings: ONE, ALL, QUORUM).

The 'Raw' measurement type, used in our experiments, requires further manual analysis of the benchmarked data. Though, it also provides a great flexibility for a posterior analysis and allows us to get important insights rarely discussed by other researchers. In particular, we noticed that the *cold start* phenomenon can have a significant effect on the results of the Cassandra performance analysis. This phenomenon exhibits itself through the initial period of low performance observed at the beginning of each read and write tests (see Figs. 2-3). In general, its duration depends on the database size, available RAM, intensity of read and write requests and their distribution and other factors. In all our experiments this period lasts approximately 800-1000 milliseconds. This phenomenon is explained by the fact that Cassandra uses three layers of data store: *memtable* (stored in RAM and periodically flushed to disk), *commit log* and *SSTable* (both are stored on disk). If the requested row is not in *memtable*, a read needs to look-up in all the *SSTable* files on disk to load data to *memtable*. In addition, Cassandra also supports integrated cacheing and distributes cache data around the cluster. Thus, during the cold start period Cassandra reads data from *SSTables* to *memtables* and warms up cache.

This period is taken out of consideration in our further statistical analysis. Otherwise, the average performance estimates would be significantly biased. For instance, in our experiments the delays measured during the cold start are on average 5-8 times longer than the ones measured during the rest of time.

IV. Data Analysys. Interdependency between Performance and Consistency

A. Read/Write Latency and Throughput Statistics

Results of the Cassandra performance benchmarking are summarised in Tables I and II and depicted in Figs. 4 and 5.

TABLE I. Cassandra READ Performance Statistics

Threads	Average latency, us					Coefficient of variation, %			Average throughput, ops/s				
	ONE	QUORUM	*(slowdown, % of ONE)*	ALL	*(slowdown, % of ONE)*	ONE	QUORUM	ALL	ONE	QUORUM	*(slowdown, % of ONE)*	ALL	*(slowdown, % of ONE)*
10	8120	8150	(+1%)	8311	(+2%)	17%	17%	15%	1136	1023	(+11%)	959	(+19%)
50	12207	13077	(+7%)	14732	(+21%)	29%	27%	21%	3525	3295	(+7%)	3181	(+11%)
100	15768	18139	(+15%)	21428	(+36%)	32%	30%	24%	6323	5489	(+15%)	5180	(+22%)
200	21853	26350	(+21%)	29218	(+34%)	56%	43%	38%	7959	7022	(+13%)	6271	(+27%)
300	31038	35996	(+16%)	41326	(+33%)	63%	48%	45%	9025	7815	(+15%)	7011	(+29%)
400	38928	48054	(+23%)	52921	(+36%)	58%	49%	44%	9561	7998	(+20%)	7313	(+31%)
500	49931	59799	(+20%)	65569	(+31%)	54%	46%	40%	9723	8173	(+19%)	7438	(+31%)
600	56433	72083	(+28%)	77215	(+37%)	50%	42%	39%	10221	8496	(+20%)	7586	(+35%)
700	69527	79919	(+15%)	84427	(+21%)	49%	39%	37%	9899	8567	(+16%)	7956	(+24%)
800	74766	87445	(+17%)	93092	(+25%)	47%	39%	35%	10487	9041	(+16%)	8322	(+26%)
900	89479	98086	(+10%)	107238	(+20%)	48%	39%	36%	10248	8906	(+15%)	8281	(+24%)
1000	96854	106762	(+10%)	117367	(+21%)	45%	41%	38%	10508	9072	(+16%)	8398	(+25%)
		Average::	(+15%)		(+26%)	46%	38%	34%			(+15%)		(+25%)

TABLE II. Cassandra WRITE Performance Statistics

Threads	Average latency, us					Coefficient of variation, %			Average throughput, ops/s				
	ONE	QUORUM	*(slowdown, % of ONE)*	ALL	*(slowdown, % of ONE)*	ONE	QUORUM	ALL	ONE	QUORUM	*(slowdown, % of ONE)*	ALL	*(slowdown, % of ONE)*
10	8941	8988	(+1%)	9116	(+2%)	20%	18%	16%	1066	1053	(+1%)	921	(+4%)
50	11198	11317	(+1%)	12591	(+12%)	28%	28%	26%	4043	3861	(+5%)	3720	(+9%)
100	14550	14747	(+1%)	16235	(+12%)	37%	32%	28%	6384	6228	(+3%)	5937	(+8%)
200	19825	20464	(+3%)	22064	(+11%)	51%	42%	32%	8803	8361	(+5%)	8074	(+9%)
300	24119	26078	(+8%)	27458	(+14%)	68%	58%	40%	10679	10334	(+3%)	9871	(+8%)
400	29944	33338	(+11%)	35319	(+18%)	61%	55%	49%	12041	11294	(+7%)	10380	(+16%)
500	36831	38784	(+5%)	41364	(+12%)	60%	53%	46%	12686	12200	(+4%)	11530	(+10%)
600	41412	44240	(+7%)	46963	(+13%)	58%	47%	45%	13444	12548	(+7%)	12054	(+12%)
700	48256	53276	(+10%)	55192	(+14%)	56%	51%	46%	13533	12714	(+6%)	12287	(+10%)
800	55629	61712	(+11%)	64856	(+17%)	54%	51%	47%	13369	12572	(+6%)	11913	(+12%)
900	61410	67329	(+10%)	69615	(+13%)	53%	49%	44%	13552	13051	(+4%)	12568	(+8%)
1000	65254	72726	(+11%)	78333	(+20%)	51%	49%	46%	13428	12985	(+3%)	12141	(+11%)
		Average::	(+7%)		(+13%)	50%	44%	39%			(+5%)		(+10%)

Fig. 4 shows that the average delay for both read and write requests increases almost linearly as the number of threads increases. The average values have been computed over a thousand of requests sent within each thread.

A coefficient of variation (CV) is a ratio between the delay standard deviation and its average value. It is used as the measure of uncertainty. This uncertainty is caused by noise (coming from the underlying platforms and technologies) and natural uncertainty and variability which is intrinsic to a cloud environment and the Internet [24, 25, 26]. The CV value (as it is shown in Tables I and II) depends on the workload (the higher the workload, the higher the latency variation) and consistency settings (the stronger the level of consistency, the lower latency variation). It varies between 34 and 50% on average, which does not affect the statistical significance of the reported average latency.

When Cassandra is configured to provide consistency level ONE, the latency of both read and write operations is lower (by 13% and 26% respectively) than the average response time of the ALL consistency setting. The QUORUM setting demonstrates a rational balance between delays and data consistency. Besides, our results confirm the claim that Cassandra has very high write speed especially under the heavy workload. Indeed, write operations are almost 25% faster on average than read requests independently of consistency settings. However, reads are slightly over-performing writes when a number of concurrent requests is below 10.

Cassandra writes executed under the ONE consistency level reach the maximum throughput of 13552 requests per second. For the QUORUM and ALL consistency settings it fluctuates around 13000 and 12500 requests per second. The maximum throughput of read operations is lower by 21%, 33% and 38% correspondingly.

A combination of average delay and average throughput columns of Tables I and II allow us to analyse how the average read and write delays depend on the current workload. When the workload reaches the maximum Cassandra throughput, delays increase in exponential progression (see Fig. 5). Figs. 4 and 5 clearly show performance benefits offered by weaker consistency settings in case of the heavy workload. It is also shown that the system is saturated with around 800 threads and delays become highly volatile when Cassandra operates close to its maximal throughput.

B. Theoretical Regressions of the Cassandra Latency

Benchmarking results reported in Tables I and II are a discrete set of measured values. They do not allow system developers to precisely estimate the database latency throughout a range of possible workloads. A regression function estimated from the experimental data (see Fig. 4) will effectively solve this problem. Table III reports the R-squared values (often referred to as the goodness-of-fit [27]) estimating extrapolation accuracy of different regression functions. It shows that the polynomial regression of the forth order (2-7) fits the experimental statistics with the high accuracy.

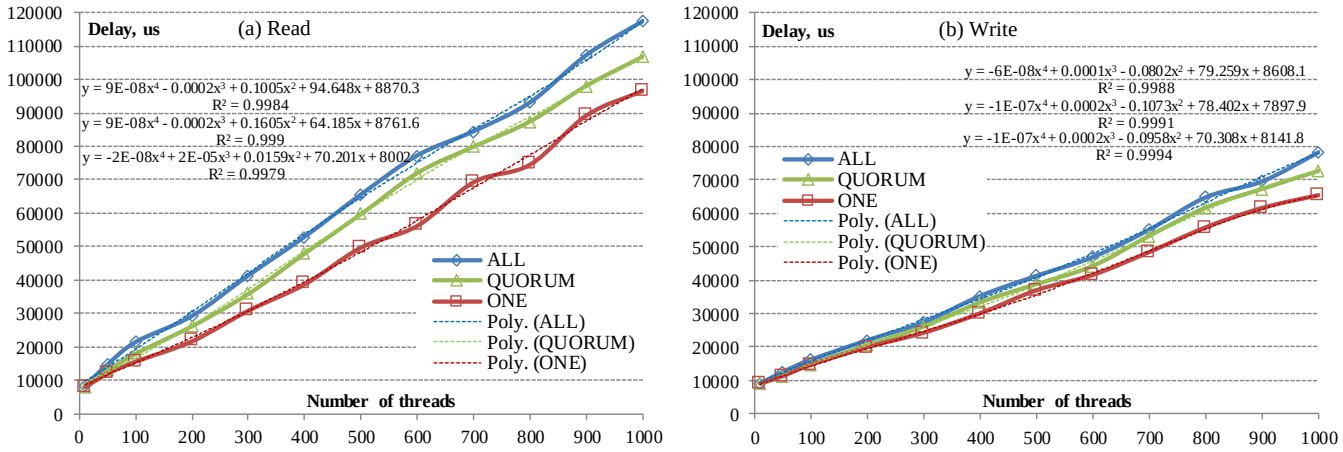

Fig. 4. Average Cassandra delay depending on the current workload: (a) reads; (b) writes.

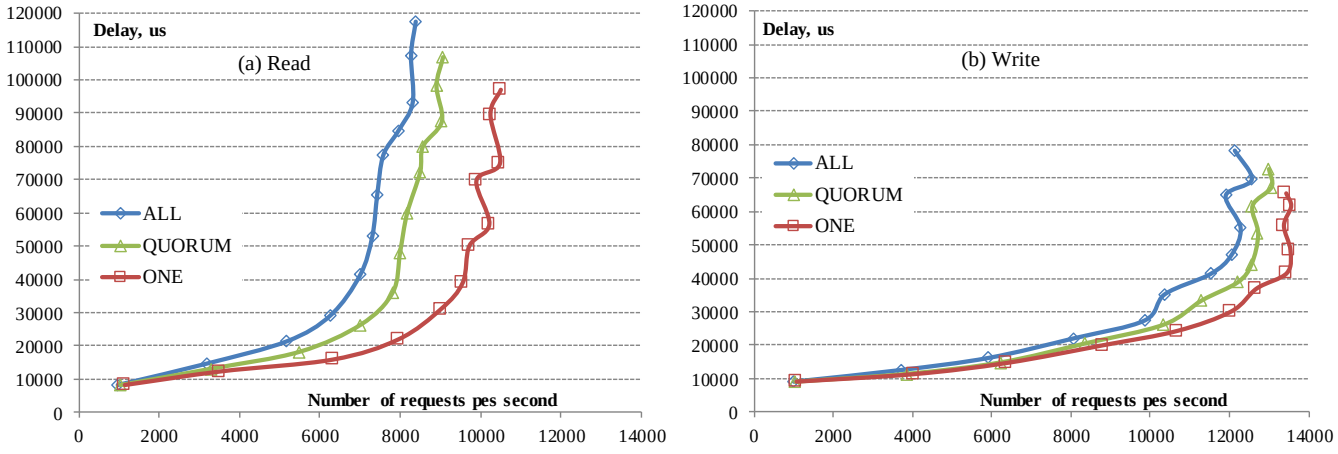

Fig. 5. Average Cassandra delay vs average throughput: (a) reads; (b) writes.

$$y^{Read}_{ALL}(x)=9E\text{-}08x^4 - 0.0002x^3 + 0.1005x^2 + 94.648x + 8870.3 \quad (2)$$

$$y^{Read}_{QUORUM}(x)=-9E\text{-}08x^4 -0.0002x^3 + 0.1605x^2 + 64.185x + 8761.6 \quad (3)$$

$$y^{Read}_{ONE}(x)=-2E\text{-}08x^4 + 2E\text{-}05x^3 + 0.0159x^2 + 70.201x + 8002 \quad (4)$$

$$y^{Write}_{ALL}(x)=-6E\text{-}08x^4 + 0.0001x^3 - 0.0802x^2 + 79.259x + 8608.1 \quad (5)$$

$$y^{Write}_{QUORUM}(x)=-1E\text{-}07x^4 + 0.0002x^3 - 0.1073x^2 + 78.402x + 7897.9 \quad (6)$$

$$y^{Write}_{ONE}(x)=-1E\text{-}07x^4 + 0.0002x^3 - 0.0958x^2 + 70.308x + 8141.8 \quad (7)$$

where y^{Read}_{ALL}, y^{Read}_{QUORUM}, y^{Read}_{ONE}, y^{Write}_{ALL}, y^{Write}_{QUORUM}, y^{Write}_{ONE} – Cassandra read/update response time for different consistency settings [us]; x – the number of threads (e.g. concurrent requests).

TABLE III. GOODNESS-OF-FIT FOR READ/WRITE REGRESSIONS

		Polynomial regression			Linear regression
		order=2	*order=3*	*order=4*	
Read statistics	ALL	0.9982	0.9982	**0.9984**	0.9979
	QUORUM	0.9981	0.9988	**0.9990**	0.9976
	ONE	0.9978	0.9979	**0.9980**	0.9952
Write statistics	ALL	0.9985	0.9985	**0.9988**	0.9970
	QUORUM	0.9981	0.9983	**0.9991**	0.9972
	ONE	0.9982	0.9986	**0.9994**	0.9978

Obviously, regression functions (2-7) and their coefficients are unique for our experimental setup and will not exactly suit other installations. System developers performing predictive modelling and forecasting system performance will need to find equations which theoretically fit their own benchmarking results using a variety of tools and APIs. If needed, more sophisticated regression techniques, like multivariate adaptive regression splines, support vector regression or artificial neural networks can be also applied [12].

V. FINDING THE OPTIMAL SETTINGS GUARANTYING THE STRONG DATA CONSISTENCY

As it was discussed in Section II.B Cassandra can guarantee the strong data consistency model if a sum of replicas written and read is higher than the replication factor. It means that for a three-replicated system (which is a default standard for many large-scale distributed systems including Facebook, Twitter, etc.) there are 6 possible read/write consistency settings guaranteeing the strong data consistency:

1) *'Read ONE – Write ALL' (1R-3W);*
2) *'Read QUORUM – Write QUORUM' (2R-2W);*
3) *'Read ALL – Write ONE' (3R-1W);*
4) *'Read QUORUM – Write ALL' (2R-3W);*
5) *'Read ALL– Write QUORUM' (3R-2W);*
6) *'Read ALL – Write ALL' (3R-3W).*

Besides, the two settings: 'Read ONE – Write QUORUM' (1R-2W) and 'Read QUORUM – Write ONE' (2R-1W) provide 66.6% consistency confidence (e.g. a probability that a read request returns the most recent data). Finally, the 'Read ONE – Write ONE' (1R-1W) setting can guarantee only the 33.3% consistency confidence.

If a system developer would like to ensure that a read operation always reflects the most recent update he/she can opt for one of the first six settings. However, our experiments clearly show that the fewer replicas are invoked the faster Cassandra performs read/write operations. Thus, in practice one should choose between the three following settings: 1R-3W, 2R-2W and 3R-1W.

As all three settings guarantee the strong consistency a system developer could be interested in choosing one providing the minimal response delay on average. In turn, the response delay and the Cassandra throughput depend on the current workload and the ratio between read/write requests.

Using regression functions (2-7) we can predict the average Cassandra latency under the mixed read-write workload:

$$y_{1R\text{-}3W}(x) = P_{Read} \cdot y^{Read}_{ONE}(x) + P_{Write} \cdot y^{Write}_{ALL}(x) \quad (8)$$

$$y_{2R\text{-}2W}(x) = P_{Read} \cdot y^{Read}_{QUORUM}(x) + P_{Write} \cdot y^{Write}_{QUORUM}(x) \quad (9)$$

$$y_{3R\text{-}1W}(x) = P_{Read} \cdot y^{Read}_{ALL}(x) + P_{Write} \cdot y^{Write}_{ONE}(x) \quad (10)$$

where P_{Read}, P_{Write} – probabilities of read/write requests, $P_{Read} + P_{Write} = 1$.

Table IV provides some estimates of the Cassandra latency for different settings guaranteeing the strong consistency under a mixed read/write workload using (2-7) and (8-10). The Cassandra database is known for extremely fast writes/updates. Thus, one might decide to use 1R-3W as the best setting, among others (e.g. 2R-2W, 3R-1W), which guaranty the strong consistency. However, as it follows from Table IV, the 1R-3W setting does not provide the lowest response time in all possible scenarios. For instance, if a probability of reads is less than 0.5 (50%), the 2R-2W consistency setting provides the lowest delay for workloads less than 50 threads. However, with the increase of the percentage of write requests (higher than 50%) the 1R-3W setting becomes optimal independently on the current workload. Increasing the number of requests per second and the percentage of read requests make the 2R-2W and especially the 3R-1W setups very inefficient demonstrating the exponential grow of the Cassandra latency. However, the 3R-1W setup still provide the lowest delay in heavy 'write mostly' workloads when the percentage of read requests is less than 25%.

The 3D surface plot shown in Fig. 6 demonstrates the domains in the input workload where the particular consistency setting provides the best performance. It is shown that opting for 2R-2W and 3R-1W in certain scenarios would allow us to improve the Cassandra performance (up to 14% on average in our case – see Fig. 7). This information can be extremely useful for system developers allowing them to dynamically change consistency settings of read and write requests in an optimal way still guaranteeing the strong data consistency.

TABLE IV. ESTIMATED CASSANDRA DELAY UNDER DIFFERENT WORKLOADS DEPENDING ON READ/WRITE RATIO

Threads	Read/Write = 10/90%			Read/Write = 30/70%			Read/Write = 50/50%			Read/Write = 90/10%		
	1R-3W	*2R-2W*	*3R-1W*	*1R-3W*	*2R-2W*	*3R-1W*	*1R-3W*	*2R-WU*	*3R-1W*	*1R-3W*	*2R-2W*	*3R-1W*
10	9324	**8746**	8935	9187	**8896**	9133	9049	**9045**	9331	**8774**	9345	9728
50	12304	**11651**	11679	12137	**11806**	12157	11970	**11960**	12636	**11637**	12269	13593
100	15794	15027	**14863**	15661	**15372**	15820	**15529**	15717	16777	**15264**	16407	18691
200	22281	21286	**20741**	**22397**	22433	22917	**22512**	23580	25093	**22743**	25874	29446
300	28532	27402	**26428**	**29051**	29622	29938	**29570**	31842	33447	**30609**	36283	40466
400	34928	33804	**32322**	**35926**	37104	37075	**36924**	40405	41828	**38921**	47006	51335
500	41709	40714	**38628**	**43204**	44930	44425	**44700**	49146	50222	**47690**	57578	61815
600	48976	48149	**45371**	**50952**	53035	51989	**52928**	57921	58607	**56879**	67694	71843
700	56686	55914	**52385**	**59115**	61239	59672	**61544**	66564	66959	**66403**	77214	81534
800	64655	63610	**59319**	67523	69247	**67285**	**70392**	74885	75250	**76128**	86160	91182
900	72561	70628	**65636**	75888	76650	**74541**	**79216**	82672	83447	**85872**	94716	101258
1000	79937	76154	**70612**	83803	82922	**81061**	**87670**	89691	91510	**95403**	103228	112407

*delays are measured in [us]; **the minimal values are underlined

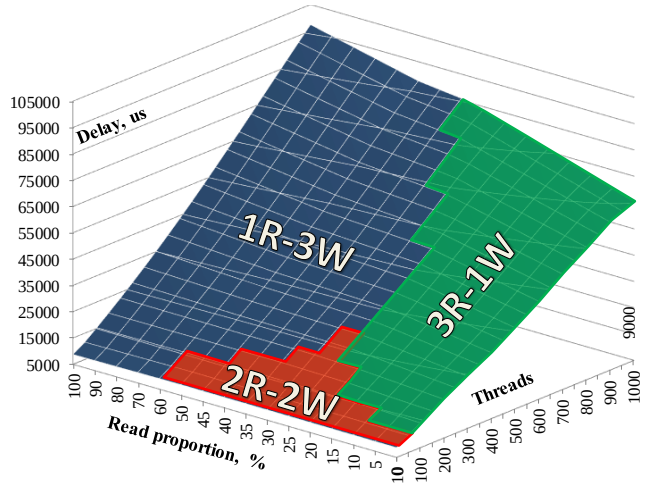

Fig. 6. Workload domains with the optimal consistency settings.

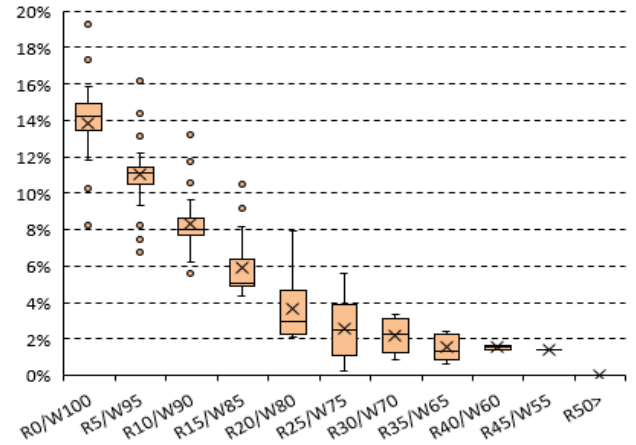

Fig. 7. Box-and-whisker diagrams showing Cassandra average speedup (latency decrease as compared to 1R-3W) due to optimal coordination of consistency settings depending on a read/write ratio.

VI. Experimental-based Methodology for Optimal Coordination of Consistency Settings

One might note that the data reported and the regression functions used in the previous sections are unique for our experimental setup and might not exactly match other installations. This is generally true. It is obvious that the Cassandra latency depends on many factors including the size and structure of the column family, used hardware, number of nodes and their geographical distribution, etc. In this section we generalize the experimental data reported by proposing a methodology to be used by system engineers for predicting the Cassandra latency and coordinating consistency settings at run time for read and write requests in an optimal manner.

A. Methodology

The methodology employs a benchmarking approach to quantify the Cassandra latency and throughput. The benchmarking here aims at estimating the system performance in order to find how efficient the system can serve the certain mixed workload when using different consistency settings. The methodology consists of the following steps (steps 1-5 can be performed once as a part of system load testing; step 6 should be performed at run-time during system operation):

1) Deploying and running a Cassandra database in a real production environment.

2) Modifying the YCSB workloads to execute application-specific read and write queries. This helps evaluate the Cassandra performance in the realistic application scenarios.

3) Benchmarking the Cassandra database under different workloads (threads per second) with different consistency settings following the benchmarking scenario described in Sections III.A-B.

4) Finding regression functions that accurately interpolate the average read/write latency measured experimentally depending on the workload for different consistency settings (see Section IV.B).

5) Identifying the optimal consistency settings by using functions (8)–(10) to provide the minimum Cassandra latency depending on the workload and the ratio of read and write requests (e.g. workload mix) as described in Section V. As a result, system developers are able to identify the workload domains with the optimal consistency settings (e.g. see Fig. 6).

6) Monitoring the current workload and the read/write ratio during system operation and setting the optimal consistency taking into account the workload domains identified at the previous step.

The proposed methodology enables a run-time optimization of consistency settings to achieve the maximal Cassandra performance and still guarantee the strong consistency.

B. Verification

To verify the proposed methodology we benchmarked the Cassandra performance under mixed read/write workloads (Read/Write = 10/90%, 30/70%, 50/50% and 90/10%) using the same methodology described in Section III.A-B. Obtained experimental results were compared with the estimated data reported in Table IV. Table V shows a deviation between experimentally measured and estimated (see Table IV) delays. Cells with consistency settings that provided the minimal latency among {1R-3W, 2R-2W and 1R-3W} are underlined in the table. It is shown that accuracy between experimental and estimated data are considerably high. A deviation never exceeds 17% (the worst case: Read/Write = 90/10%; threads=10; 3R-1W) and is reducing with the increase of a number of threads.

TABLE V. A Deviation Between Estimated and Experimentally Measured Cassandra Delays under Different Workloads Depending on Read/Write Ratio

Threads	Read/Write = 10/90%			Read/Write = 30/70%			Read/Write = 50/50%			Read/Write = 90/10%		
	1R-3W	*2R-2W*	*3R-1W*	*1R-3W*	*2R-2W*	*3R-1W*	*1R-3W*	*2R-WU*	*3R-1W*	*1R-3W*	*2R-2W*	*3R-1W*
10	-3.4%	1.8%	-0.6%	-4.2%	-1.8%	-4.3%	-5.0%	-5.6%	-8.2%	-6.7%	-13.5%	-16.2%
50	2.0%	-1.4%	-1.1%	2.7%	0.3%	0.8%	3.5%	1.9%	2.5%	5.0%	4.9%	5.5%
100	2.4%	0.4%	2.5%	2.7%	2.5%	4.8%	3.0%	4.4%	6.7%	3.5%	7.8%	9.9%
200	-1.1%	-1.1%	0.1%	-1.8%	-0.9%	-1.2%	-2.5%	-0.7%	-2.3%	-4.0%	-0.4%	-4.1%
300	-2.6%	-1.2%	-2.3%	-1.8%	-2.0%	-2.2%	-1.1%	-2.6%	-2.2%	0.2%	-3.7%	-2.2%
400	2.1%	2.9%	-0.2%	1.3%	1.7%	-0.6%	0.5%	0.7%	-1.0%	-0.9%	-0.9%	-1.4%
500	1.2%	0.4%	2.7%	1.7%	0.4%	2.3%	2.1%	0.3%	1.9%	2.8%	0.2%	1.4%
600	-2.2%	-2.4%	-0.8%	-2.3%	-0.8%	0.3%	-2.4%	0.4%	1.2%	-2.5%	2.3%	2.4%
700	-0.1%	0.0%	-1.0%	0.6%	0.0%	-1.0%	1.3%	0.0%	-0.9%	2.5%	0.1%	-0.9%
800	1.8%	1.1%	0.1%	0.5%	0.3%	-0.6%	-0.8%	-0.4%	-1.2%	-3.2%	-1.5%	-2.1%
900	-1.3%	-0.3%	0.5%	-0.4%	-0.1%	0.8%	0.4%	0.0%	1.0%	1.9%	0.3%	1.4%
1000	0.3%	0.0%	-0.2%	0.1%	0.0%	-0.2%	-0.1%	0.1%	-0.2%	-0.4%	0.1%	-0.2%

*deviations for the measured minimum delays are underlined

Finally, by matching the cells with the underlined values in Tables IV and V one should note that the proposed methodology suggests optimal consistency settings in 92%.

VII. Conclusion and Lessons Learnt

Our work experimentally investigates the interplay between different consistency settings and performance of the Cassandra NoSQL database. This is an important part of the fundamental trade-off between Consistency, Availability and Partition tolerance which is in the very nature of globally-distributed systems and large-scale replicated data storages.

The reported results show that used consistency settings can significantly affect the Cassandra response time and throughput that have to be accounted during system design and operation. The strong data consistency settings can increase database latency by 26% and degrade its throughput by 25% on average for read requests and by 13% and 10% for write requests correspondingly.

The Cassandra database offers developers a unique opportunity to tune the consistency setting for each read or write request. Besides, it is possible to guarantee the strong data consistency by coordinating consistency settings for read and write requests to ensure that the sum of nodes written and read is greater than the replication factor. Developers of real Big-Data applications where Cassandra is used as a NoSQL storage are advised to benchmark the performance of different consistency settings under different workloads and for different ratios between read and write requests. This will allow them to identify the domains in the space of the input workload where the certain consistency setting provides the minimum latency.

One of our major findings is the fact that the optimal consistency settings maximizing the Cassandra performance significantly depend on the current workload and a ratio of read and write requests. They confirm our claim that none of consistency settings always guaranties the minimum latency.

There could be no “cleanly” defined workload mixes which approximate the operational system workloads to make the best off-line decisions. The real workload mix can evolve and change over time impacting the desirable Cassandra settings. The proposed methodology aims at choosing the optimal consistency setting dynamically at run-time by monitoring the current workload mix and making use of the benchmarking results collected offline during system load testing.

References

[1] E. Evans, "NoSQL 2009," 12 May 2009. [Online]. Available: http://blog.sym-link.com/2009/05/12/nosql_2009.html.

[2] D. Pritchett, "Base: An Acid Alternative," *ACM Queue*, vol. 6, no. 3, pp. 48-55, 2008.

[3] E. Brewer, "Towards Robust Distributed Systems," in *19th Annual ACM Symposium on Principles of Distributed Computing*, Portland, USA, 2000.

[4] S. Gilbert and N. Lynch, "Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51-59, 2002.

[5] Github, "Benchmarking Cassandra and other NoSQL databases with YCSB," [Online]. Available: https://github.com/cloudius-systems/osv/wiki/Benchmarking-Cassandra-and-other-NoSQL-databases-with-YCSB.

[6] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *1st ACM Symposium on Cloud Computing*, Indianapolis, Indiana, USA, 2010.

[7] R. Hecht and S. Jablonski, "NoSQL Evaluation. A Use Case Oriented Survey," in *IEEE International Conference on Cloud and Service Computing*, Washington, USA, 2011.

[8] V. Abramova, J. Bernardino and P. Furtado, "Testing Cloud Benchmark Scalability with Cassandra," in *IEEE 10th World Congress on Services*, Anchorage, USA, 2014.

[9] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham and C. Matser, "Performance Evaluation of NoSQL Databases: A Case Study," in *1st ACM/SPEC International Workshop on Performance Analysis of Big Data Systems*, Austin, USA, 2015.

[10] G. Haughian, R. Osman and W. Knottenbelt, "Benchmarking Replication in Cassandra and MongoDB NoSQL Datastores," in *27th International Conference on Database and Expert Systems Applications*, Porto, Portugal, 2016.

[11] V. A. Farias, F. R. Sousa, J. G. R. Maia, J. P. P. Gomes and J. C. Machado, "Regression based performance modeling and provisioning for NoSQL cloud databases," *Future Generation Computer Systems*, vol. 79, p. 72–81, 2018.

[12] F. Karniavoura and K. Magoutis, "A measurement-based approach to performance prediction in NoSQL systems," in *25th IEEE Int. Symposium on the Modeling, Analysis, and Simulation of Computer and Telecom. Systems (MASCOTS'07)*, Banff, Canada, 2017.

[13] F. Cruz, F. Maia, M. Matos, R. Oliveira, J. Paulo, J. Pereira and R. Vilaca, "Resource usage prediction in distributed key-value datastores," in *IFIP Distributed Applications and Interoperable Systems (DAIS'2017)*, Heraklion, Crete, 2017.

[14] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23-29, 2012.

[15] R. Guerraoui, M. Pavlovic and D. Seredinschi, "Trade-offs in replicated systems," *IEEE Bulletin of the Technical Committee on Data Engineering*, vol. 39, no. 1, pp. 14-26, 2016.

[16] D. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design," *IEEE Computer*, vol. 45, no. 2, pp. 37-42, 2012.

[17] O. Tarasyuk, A. Gorbenko and A. Romanovsky, "The Impact of Consistency on System Latency in Fault Tolerant Internet Computing," in *Distributed Applications and Interoperable Systems, LNCS 9038*, Berlin, Springer, 2015, pp. 179-192.

[18] A. Fekete and K. Ramamritham, "Consistency Models for Replicated Data," in *Replication*, vol. LNCS 5959, B. Charron-Bost, F. Pedone and A. Schiper, Eds., Berlin, Springer-Verlag, 2010, pp. 1-17.

[19] S. Burckhardt, "Principles of Eventual Consistency," *Foundations and Trends Programming Languages*, vol. 1, no. 1-2, pp. 1-150, 2014.

[20] DataStax, Inc., "Apache Cassandra 2.1 for DSE. About data consistency," 14 February 2018. [Online]. Available: https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dmlAboutDataConsistency.html.

[21] N. Neeraj, Mastering Apache Cassandra, Birmingham: Packt Publishing Ltd. , 2013.

[22] DataStax, Inc., "Apache Cassandra 2.1. Configuring data consistency," 14 February 2018. [Online]. Available: https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dml_config_consistency_c.html.

[23] B. Cooper, "Running a Workload," 2 Jul 2013. [Online]. Available: https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload.

[24] Y. Chen, A. Gorbenko, A. Romanovsky and V. Kharchenko, "Measuring and Dealing with the Uncertainty of the SOA Solutions," in *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, V. Cardellini, Ed., Hershey, USA, IGI Global, 2011, p. 265–294.

[25] A. Gorbenko, A. Romanovsky, O. Tarasyuk, V. Kharchenko and S. Mamutov, "Exploring Uncertainty of Delays as a Factor in End-to-End Cloud Response Time," in *9th European Dependable Computing Conference (EDCC'2012)*, Sibiu, Romania, 2012.

[26] H. Mezni, S. Aridhi and A. Hadjali, "The uncertain cloud: State of the art and research challenges," *International Journal of Approximate Reasoning*, vol. 103, pp. 139-151, 2018.

[27] J. L. Devore, Probability and Statistics for Engineering and the Sciences, 8th Edition, Boston (USA): Cengage Learning, 2011.