ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

UDC 004.658.051 **doi: 10.32620/reks.2020.4.09**

A. GORBENKO[1,2]**, O. TARASYUK**[3]

[1] ***Leeds Beckett University, UK***
[2] ***National Aerospace University "Kharkiv Aviation Institute", Ukraine***
[3] ***Odessa Technological University STEP, Ukraine***

EXPLORING TIMEOUT AS A PERFORMANCE AND AVAILABILITY FACTOR OF DISTRIBUTED REPLICATED DATABASE SYSTEMS

A concept of distributed replicated data storages like Cassandra, HBase, MongoDB has been proposed to effectively manage the Big Data sets whose volume, velocity, and variability are difficult to deal with by using the traditional Relational Database Management Systems. Trade-offs between consistency, availability, partition tolerance, and latency are intrinsic to such systems. Although relations between these properties have been previously identified by the well-known CAP theorem in qualitative terms, it is still necessary to quantify how different consistency and timeout settings affect system latency. The paper reports results of Cassandra's performance evaluation using the YCSB benchmark and experimentally demonstrates how to read latency depends on the consistency settings and the current database workload. These results clearly show that stronger data consistency increases system latency, which is in line with the qualitative implication of the CAP theorem. Moreover, Cassandra latency and its variation considerably depend on the system workload. The distributed nature of such a system does not always guarantee that the client receives a response from the database within a finite time. If this happens, it causes so-called timing failures when the response is received too late or is not received at all. In the paper, we also consider the role of the application timeout which is the fundamental part of all distributed fault tolerance mechanisms working over the Internet and used as the main error detection mechanism here. The role of the application timeout as the main determinant in the interplay between system availability and responsiveness is also examined in the paper. It is quantitatively shown how different timeout settings could affect system availability and the average servicing and waiting time. Although many modern distributed systems including Cassandra use static timeouts it was shown that the most promising approach is to set timeouts dynamically at run time to balance performance, availability and improve the efficiency of the fault-tolerance mechanisms.

Keywords: *timeout; NoSQL; distributed databases; replication; performance benchmarking; consistency; availability; latency; trade-off.*

Introduction

Distributed data storages have become the standard platform and a major industrial technology for dealing with enormous data growth. They are now widely used in different application domains, including distributed Internet applications, social networks and media, critical infrastructures, business-critical systems, IoT and industrial systems. A new generation of such databases are called NoSQL (Not Only SQL or NO SQL) [1]. They are designed to provide horizontal scalability and employ Internet-scale replication to guaranty high availability, throughput and low latency.

A concept of distributed data storages has been proposed to effectively manage the Big Data sets whose volume, velocity and variability are difficult to deal with by using the traditional Relational Database Management Systems. Most NoSQL databases 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. They are also subject to the tradeoff between Consistency, Availability, and Partition tolerance (CAP).

The CAP theorem [3], first appeared in 1998-1999, declares that the only two of the three properties can be preserved at once in distributed replicated systems. Gilbert and Lynch [4] consider the CAP theorem as a particular case of a more general trade-off between consistency and availability in unreliable distributed systems propagating updates eventually over time. However, a tradeoff between availability and latency is less studied. There have been a number of studies, e.g. [5 - 8], 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 major focus of those works is to compare and select the best NoSQL databases based on performance measures. However, reported results show that performance of different NoSQL databases significantly depends on the use case scenario, deployment conditions, current workload and

database settings. Thus, there is no NoSQL database that always outperforms the others. Other recent related works, such as [9 - 11], have investigated measurement-based performance prediction of NoSQL data stores. However, the studies, mentioned above, do not investigate an interdependency between availability and performance and do not study how time-out settings affect database latency.

The aim of this work is to experimentally evaluate a trade-off between availability and latency, which is in the very nature of NoSQL databases, and to study how timeout settings can be used to interplay between them.

1. The role of the application timeout as the main performance and availability factor

Most error recovery and fault-tolerance techniques depend on the time-out setup. In particular, setting appropriate time-outs is key to improving many distributed systems' performance and dependability. However, researchers have focused mainly on optimizing timeouts used by communication protocols [12, 13]. They haven't examined how application level timeout settings affect performance and dependability of distributed systems. 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 a probability that each client request eventually receives a response. In many real systems, however, a response that is too late (i.e. beyond the application timeout) is treated as a failure.

For example, the failure model introduced by Avizienis, et al. in [14] distinguishes between the two main failure domains in distributed systems: (i) *timing failures* when the duration of the response delivered to the client exceeds the specified waiting time – the application timeout (i.e. the service is delivered too late), and (ii) *content failures* when the content (value) of the response deviates from implementing the system function.

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's response time [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.

Timeout settings are crucially important is distributed replicated systems. If the timeout is lower than the typical response time, a system is likely to enter the partition mode more often. On the other hand, timeout which is too high does not allow timely detect errors and failure and effectively apply fault-tolerance mechanisms. The application timeout can be considered as a bound between system availability and performance (in term of latency or response time) [16, 17]. Thus, system designers should be able to set up timeouts according to the desired system response time, also keeping in mind the choice between consistency and availability.

2. Cassandra Performance Benchmarking

In this paper we put a special focus on quantitative evaluation of one of the fundamental trade-offs between system availability and latency in distributed replicated data storages using the Cassandra NoSQL as a typical example of such system. Various industry trends suggest that Apache Cassandra is one of the top three in use today together with MongoDB and HBase [18].

2.1. Experimental setup

This section describes the performance benchmarking methodology used and reports the experimental results showing how timeout settings affect latency of the read requests for the Cassandra NoSQL database.

As a testbed we have deployed the 3-replicated Cassandra 2.1 cluster in the Amazon EC2 cloud (Fig. 1). 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 was 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).

2.2. Benchmarking methodology

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 [5]. YCSB is an open-source Java project. The YCSB framework includes six out-of-the-box workloads [5], each testing different common use case scenarios with a certain mix of reads and writes. In this paper we report experimental results corresponding to the read-only Workload C. 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

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

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

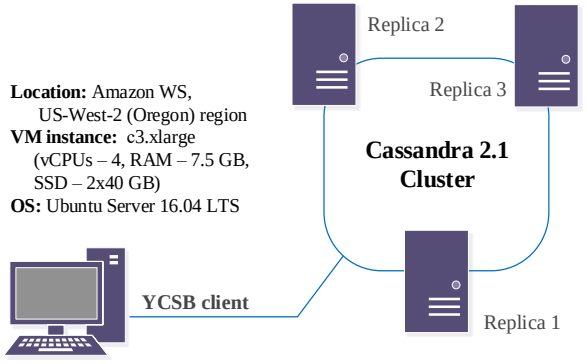

Fig. 1. Experimental setup: deployment of Cassandra NoSQL cluster

2.3. Benchmarking scenario

Some examples of general methodologies for benchmarking Cassandra and other NoSQL databases with YCSB can be found in [18]. However, unlike these and other works (e.g. [5 - 8]) studying and comparing performance of different NoSQL databases we put the focus on analysing the dynamic aspects of the Cassandra performance under different workloads (i.e. number of concurrent requests/threads) and various consistency settings (e.g. ALL, QUORUM, ONE).

Cassandra consistency model defines the number of requested replicas that must acknowledge a read (or write) operation before the most recent result is returned to the client (or the write operation is considered successful). In the paper we consider three different consistency level:

– ALL (the strongest consistency level); all replicas are queried and must respond; the most recent (based on the time stamp) read result is returned to a client;

– ONE (the weakest consistency level); only one replica is requested and must respond with the result which is returned to a client; a client can receive stalled data if the most recent updates have not been propagated to that replica;

– QUORUM (the moderate consistency level); quorum (e.g. 2-out-of-3) of replicas are queried and must respond; this level provides a compromise between data consistency and system latency.

A series of YSCB read performance tests were performed on the 3-replicated Cassandra cluster with the consistency setting set to ALL, ONE and QUORUM with a number of threads varying from 100 to 1000. The operation count within each thread was set to 1000.

3. Data analysis

3.1. Cassandra read performance

In this section we report new experimental results in addition to those discussed in our previous study [19]. Tables 1 - 3 report Cassandra read latency statistics depending on the number of requests executed in parallel (threads) and consistency settings. It is also shown that the average Cassandra latency as well as the maximum response time steadily increase with the increase of the number of threads.

Table 1

Cassandra read latency (us)
for the strong consistency setting ALL

Threads	Min	Max	Average	Std. Dev.	Ops. per second
100	3789	47818	17427	4494	5380
200	6056	100394	29217	11208	6471
300	4875	139900	41326	18638	7010
400	2319	163312	52920	23231	7312
500	7191	184161	65569	26339	7438
600	1176	233869	77215	29799	7586
700	4712	229903	84427	31298	8155
800	6703	255587	92091	32050	8521
900	2448	267868	107238	38731	8280
1000	6176	407612	117367	44185	8398

Table 2

Cassandra read latency (us)
for the consistency setting QUORUM

Threads	Min	Max	Average	Std. Dev.	Ops. per second
100	1016	67819	18138	7273	5189
200	4424	86830	26350	12764	7022
300	4892	116258	35995	15503	7814
400	5278	160904	48053	23762	7998
500	1082	179521	59799	27485	8172
600	1750	240746	72983	30551	8016
700	939	245338	79918	31542	8567
800	1225	312977	87444	33830	9040
900	3047	267239	98086	37974	9006
1000	1349	322059	110761	45804	8871

Table 3

Cassandra read latency (us)
for the weak consistency setting ONE

Threads	Min	Max	Average	Std. Dev.	Ops. per second
100	1340	67438	14268	8870	6323
200	1152	84807	20153	13394	9259
300	648	115569	31038	19683	9324
400	1668	173360	38927	22649	9660
500	761	193154	49930	26879	9723
600	1623	203336	56432	28424	10221
700	2139	203004	69526	31119	9799
800	1011	235942	74766	35047	10486
900	1504	318241	89478	44925	9848
1000	1437	347631	91853	44077	10707

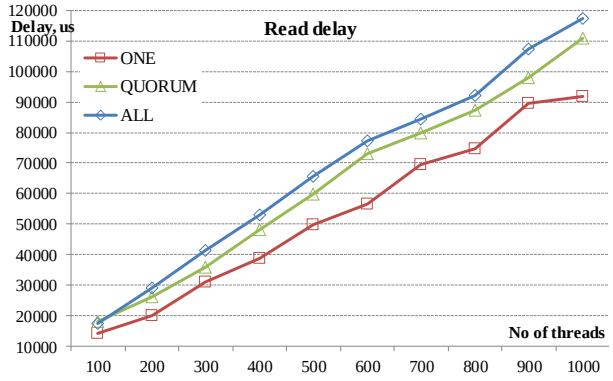

Fig. 2. Cassandra read delay depending on the number of threads

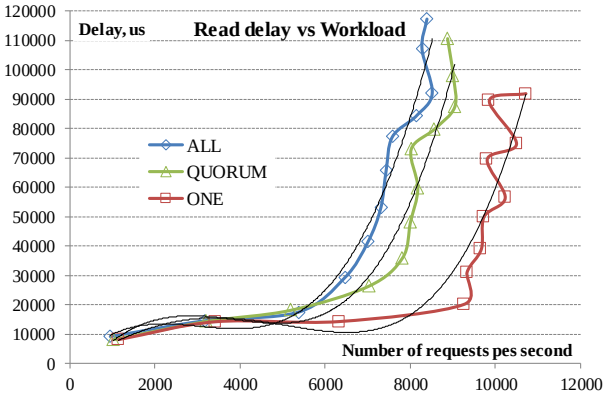

Fig. 3. Cassandra read delay depending on the database workload

It is clearly shown that the higher the level of consistency, the higher the latency of the system independently of the number of threads/database workload (see Figs. 2 - 3).

Fig. 3 shows that the system is saturated with around 800 threads on average and delays become highly volatile when Cassandra operates close to its maximal throughput. When the workload reaches the, delays increase in exponential progression. It is worth noting that Cassandra reaches the maximum throughput (approx. 1100 requests per second) when it is configured to provide the weakest consistency level ONE.

3.2. Interplay between availability and latency

Cassandra uses the following timeout values set by default: 5000 ms for read requests and 2000 ms for write requests (Cassandra is designed to perform write operation faster than read requests).

At the same time in our experiments the maximum read response time never exceeded 500 ms even for the strong consistency level ALL and the maximum number of threads. Thus, the default timeout setup is significantly higher (in 10 times!) than the worst-case execution time. Thus, Cassandra could be slow to respond to possible errors and failures that may occur during operation. At the same time, too short timeout can lead to an erroneous decision that the system has become partitioned. A general approach, widely used in communications protocols, assumes that the doubled average latency or the worst-case execution time can be set as the timeout value. However, Tables 1-3 show that the maximum response time increases with the increase if the database workload.

Moreover, as shown in Fig. 4, probability density series of Cassandra response time considerably expand with increasing database workload. On the one hand, this means that the standard deviation of Cassandra response time increases, and its latency becomes more uncertain [20]. On the other hand, it shows that timeout settings suitable for low workloads could be inadequate when the database experiences the high demand.

Fig. 4 depicts a situation if the timeout is set to 1000 ms (approx. the doubled maximum response time for the threads count 100). Red bars correspond to the situation when Cassandra would have responded after the specified timeouts. This clearly shows that the proposed timeout is too short for heavy workloads. At the same time, short timeout reduces user *servicing* and *waiting time*, as discussed in [16]. This is because the average waiting time (for all invocations, including those when a timeout is triggered) is calculated as the sum of the average time of received responses and a product of the timeout value and the probability of timeout.

Table 4

Trade-offs between system availability and latency depending on timeout settings

Threads	Timeout=100 ms			Timeout=150 ms			Timeout=200 ms		
	availability	avg. servicing time, us	avg. waiting time, us	availability	avg. servicing time, us	avg. waiting time, us	availability	avg. servicing time, us	avg. waiting time, us
100	0.99986	17428	17440	0.99986	17428	17447	1	17428	17428
200	0.99987	29213	29222	0.99993	29218	29226	1	29218	29218
300	0.97448	39598	41139	0.99996	41326	41331	1	41326	41326
400	0.94491	49232	52028	0.99846	52762	52912	1	52921	52921
500	0.88904	58806	63376	0.99633	65240	65551	1	65569	65569
600	0.80293	65908	72627	0.98600	76055	77091	0.99995	77207	77214
700	0.75438	69944	77326	0.97014	81843	83878	0.99738	84084	84387
800	0.66633	74857	83246	0.94071	87262	90982	0.99643	91591	91979
900	0.50582	77891	88817	0.85748	95656	103401	0.97554	104426	106764
1000	0.39259	77702	91246	0.78265	99083	110150	0.95472	112041	116024

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

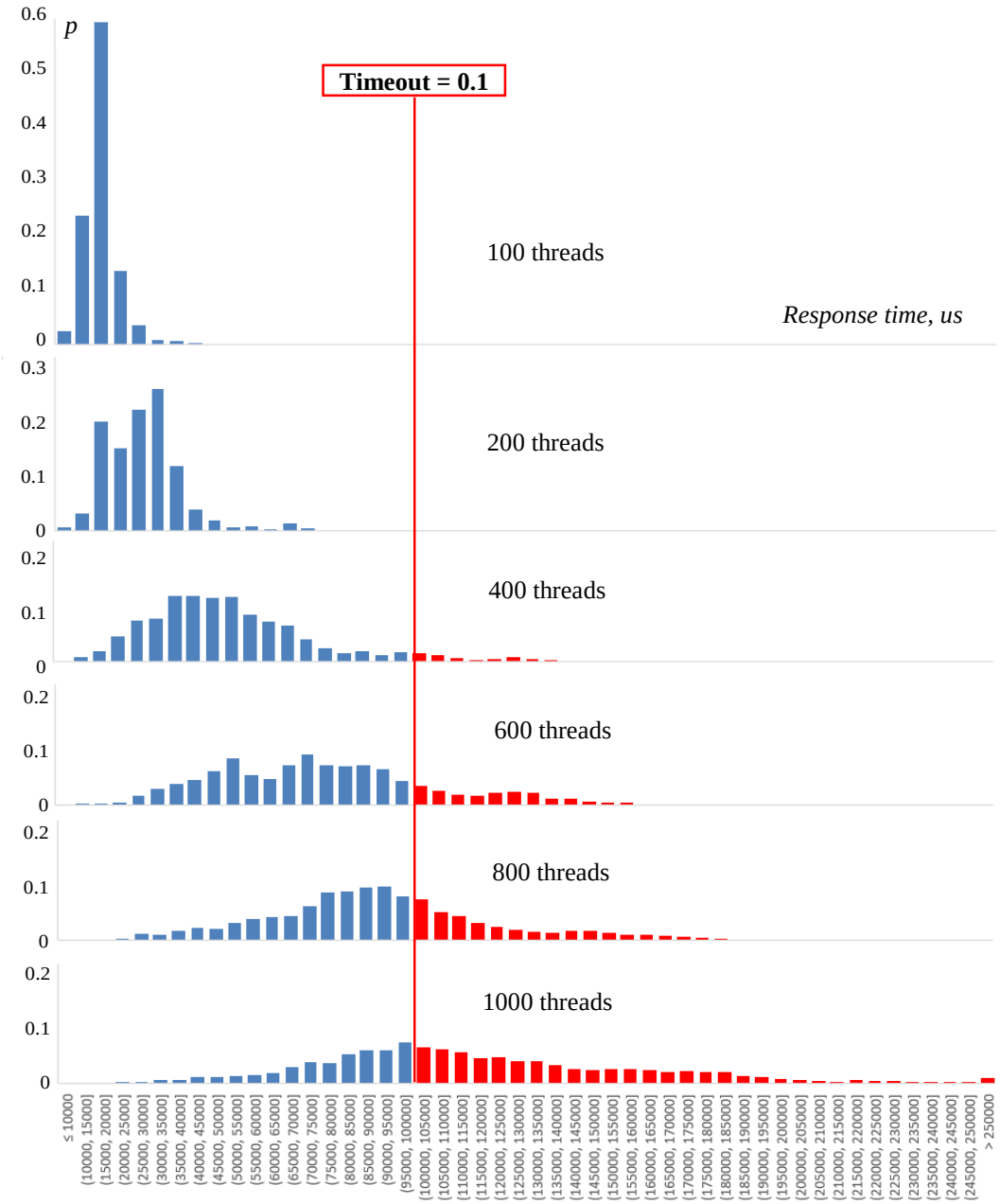

Fig. 4. Probability density series of Cassandra read ALL latency under different workloads

Ultimately, timeout could be considered as a tool to interplay between system availability and latency, as shown in Table 4. Shorter timeout reduces system latency. However, the availability of the system may also be reduced as some responses may arrive after the timeout is triggered.

Conclusions

Availability, consistency and performance of distributed database systems are tightly connected. Although these relations have been identified by the CAP theorem in qualitative terms [3, 4], it is still necessary to quantify how different timeout settings affect system latency. Understanding this trade-off is key for the effective usage of distributed databases.

In the paper we report results of Cassandra performance benchmarking and also examine the role of the application timeout as the main determinant in the interplay between system availability and responsiveness. The application timeout can be considered as a bound between system availability and performance (in term of latency or response time). Moreover, application timeout is the fundamental part of all distributed fault tolerance techniques and is used as the main error detection mechanism here. Thus, system designers should be able to set up timeouts according to the desired system response time, also keeping in mind the choice between consistency and availability.

Unfortunately, many modern distributed systems including Cassandra use static timeout settings that are often too long. This can worsen system latency and

causes ineffective failure detection and fault tolerance. Yet the most promising approach is to set timeout dynamically at run time to balance performance, availability and fault-tolerance. Our experiments show that the optimal timeout should be application specific (i.e. set depending on the database structure, volume and the most common read/write queries) and needs to be adjusted dynamically at run-time taking into account various factors, including: current system workload; number of replicas; consistency settings, etc.

References (GOST 7.1:2006)

1. Meier, A. SQL and NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management [Text] / A. Meier, M. Kaufmann. – Berlin : Springer Verlag, 2019. – 229 p.

2. Pritchett, D. Base: An Acid Alternative [Text] / D. Pritchett // ACM Queue. – 2008. – Vol. 6, No. 3. – P. 48-55.

3. Brewer, E. Towards Robust Distributed Systems [Text] / E. Brewer // Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing. – Portland, USA, 2000. – P. 7-8.

4. Gilbert, S. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services [Text] / S. Gilbert, N. Lynch // ACM SIGACT News. – 2002. – Vol. 33, No. 2. – P. 51-59.

5. Benchmarking Cloud Serving Systems with YCSB [Text] / B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears // Proceedings of the 1st ACM Symposium on Cloud Computing. – Indianapolis, USA, 2010. – P. 143-154.

6. Abramova, V. Testing Cloud Benchmark Scalability with Cassandra [Text] / V. Abramova, J. Bernardino, P. Furtado // Proceedings of the IEEE 10th World Congress on Services. – Anchorage, USA, 2014. – P. 434-441.

7. Performance Evaluation of NoSQL Databases: A Case Study [Text] / J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, C. Matser // Proceedings of the 1st ACM/SPEC Int. Workshop on Performance Analysis of Big Data Systems. – Austin, USA, 2015. – P. 5-10.

8. Haughian, G. Benchmarking Replication in Cassandra and MongoDB NoSQL Datastores [Text] / G. Haughian, R. Osman and W. Knottenbelt // Proceedings of the 27th Int. Conf. on Database and Expert Systems Applications. – Porto, Portugal, 2016. – P. 152-166.

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

10. Karniavoura, F. A measurement-based approach to performance prediction in NoSQL systems [Text] / F. Karniavoura, K. Magoutis // Proceedings of the 25th IEEE Int. Symposium on the Modeling, Analysis, and Simulation of Computer and Telecom. Systems. – Banff, Canada, 2017. – P. 255-262.

11. Resource usage prediction in distributed key-value datastores [Text] / F. Cruz, F. Maia, M. Matos, R. Oliveira, J. Paulo, J. Pereira, R. Vilaca // Proceedings of the IFIP Distributed Applications and Interoperable Systems Conf. (DAIS'2017). – Heraklion, Crete, 2017. – P. 144-159.

12. Abdelmoniem, A. M. Curbing Timeouts for TCP-Incast in Data Centers via A Cross-Layer Faster Recovery Mechanism [Text] / A. M. Abdelmoniem, B. Bensaou // Proceedings of the IEEE Conf. on Computer Communications. – Honolulu, HI, 2018. – P. 675-683.

13. Libman, L. Optimal retrial and timeout strategies for accessing network resources [Text] / L. Libman, A. Orda // IEEE/ACM Transactions on Networking. – 2002. – Vol. 10, No. 4. – P. 551-564.

14. Basic concepts and taxonomy of dependable and secure computing [Text] / A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr // IEEE Transactions on Dependable and Secure Computing. – 2004. – Vol. 1, No. 1. – P. 11-33.

15. Brewer, E. CAP twelve years later: How the "rules" have changed [Text] / E. Brewer // Computer. – 2012. – Vol. 45, No. 2. – P. 23-29.

16. Gorbenko, A. Fault tolerant internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency [Text] / A. Gorbenko, A. Romanovsky, O. Tarasyuk // Journal of Network and Computer Applications. – 2019. – Vol. 146. – P. 1-14.

17. Gorbenko, A. Time-outing Internet Services [Text] / A. Gorbenko, A. Romanovsky // IEEE Security & Privacy. – 2013. – Vol. 11, No. 2. – P. 68-71.

18. Benchmarking Cassandra and other NoSQL databases with YCSB [Electronic resource]. – Github. – Access: https://github.com/cloudius-systems/osv/wiki/Benchmarking-Cassandra-and-other-NoSQL-databases-with-YCSB. – 12.07.2020.

19. Gorbenko, A. Interplaying Cassandra NoSQL consistency and performance: A benchmarking approach [Text] / A. Gorbenko, A. Romanovsky, O. Tarasyuk // In: Communications in Computer and Information Science, Vol. 1279 / Editors: S. Bernardi, et al. – Berlin : Springer Nature, 2020. – P. 168-184.

20. The threat of uncertainty in Service-Oriented Architecture [Text] / A. Gorbenko, V. Kharchenko, O. Tarasyuk, Y. Chen, A. Romanovsky // Proceedings of the RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems. – Newcastle, UK, 2008. – P. 49-54.

References (BSI)

1. Meier, A., Kaufmann, M. *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*, Berlin, Springer Verlag Publ., 2019. 229 p.

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

3. Brewer, E. Towards Robust Distributed Systems. *Proceedings of the 19th Ann. ACM Symp. on Principles of Distributed Computing*, Portland, USA, 2000, pp. 7-8.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

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

5. Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R. Benchmarking Cloud Serving Systems with YCSB. *Proceedings of the 1st ACM Symp. on Cloud Computing*, Indianapolis, Indiana, USA, 2010, pp. 143-154.

6. Abramova, V., Bernardino, J., Furtado, P. Testing Cloud Benchmark Scalability with Cassandra. *Proceedings of the IEEE 10th World Congress on Services*. Anchorage, USA, 2014, pp. 434-441.

7. Klein, J., Gorton, I., Ernst, N., Donohoe, P., Pham, K., Matser, C. Performance Evaluation of NoSQL Databases: A Case Study. *Proceedings of the 1st ACM/SPEC Int. Workshop on Performance Analysis of Big Data Systems*, Austin, USA, 2015, pp. 5-10.

8. Haughian, G., Osman, R., Knottenbelt, W. Benchmarking Replication in Cassandra and MongoDB NoSQL Datastores. *Proceedings of the 27th Int. Conf. on Database and Expert Systems Applications*, Porto, Portugal, 2016, pp. 152-166.

9. Farias, V. A., Sousa, F. R., Maia, J. G. R., Gomes, J. P. P., Machado, J. C. Regression based performance modeling and provisioning for NoSQL cloud databases. *Future Generation Computer Systems*, 2018, vol. 79, pp. 72-81.

10. Karniavoura, F. & Magoutis, K. A measurement-based approach to performance prediction in NoSQL systems. *Proceedings of the 25th IEEE Int. Symp. on the Modeling, Analysis, and Simulation of Computer and Telecom. Systems*, Banff, Canada, 2017, pp. 255-262.

11. Cruz, F., Maia, F., Matos, M., Oliveira, R., Paulo, J., Pereira, J., Vilaca, R. Resource usage prediction in distributed key-value datastores. *Proceedings of the IFIP Distributed Applications and Interoperable Systems Conf.*, Heraklion, Crete, 2017, pp. 144-159.

12. Abdelmoniem, A. M., Bensaou, B. Curbing Timeouts for TCP-Incast in Data Centers via A Cross-Layer Faster Recovery Mechanism. *Proceedings of the IEEE Conf. on Computer Communications*, Honolulu, HI, 2018, pp. 675-683.

13. Libman, L., Orda, A. Optimal retrial and timeout strategies for accessing network resources. *IEEE/ACM Transactions on Networking*, 2002, vol. 10, no. 4, pp. 551-564.

14. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 2004, vol. 1, no. 1, pp. 11-33.

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

16. Gorbenko, A., Romanovsky, A., Tarasyuk, O. Fault tolerant internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency. *Journal of Network and Computer Applications*, 2019, vol. 146, pp. 1-14.

17. Gorbenko, A., Romanovsky, A. Time-outing Internet Services. *IEEE Security & Privacy*, 2013, vol. 11, no. 2, pp. 68-71.

18. Github, *Benchmarking Cassandra and other NoSQL databases with YCSB.* [Online]. Available at: https://github.com/cloudius-systems/osv/wiki/Benchmarking-Cassandra-and-other-NoSQL-databases-with-YCSB. (accessed 12.07.2020).

19. Gorbenko, A., Romanovsky, A., Tarasyuk, O. Interplaying Cassandra NoSQL consistency and performance: A benchmarking approach. In: S. Bernardi & e. all., eds. *Communications in Computer and Information Science.* Berlin: Springer Nature, 2020, pp. 168-184.

20. Gorbenko, A., Kharchenko, V., Tarasyuk, O., Chen, Y., Romanovsky, A. The threat of uncertainty in Service-Oriented Architecture. *Proceedings of the RISE/EFTS Joint Int. Workshop on Software Engineering for Resilient Systems*, Newcastle, 2008, pp. 49-54.

Надійшла до редакції 12.09.2020, розглянута на редколегії 16.11.2020

ДОСЛІДЖЕННЯ ТАЙМ-АУТУ ЯК ФАКТОРА ВПЛИВУ НА ПРОДУКТИВНІСТЬ І ДОСТУПНІСТЬ РОЗПОДІЛЕНИХ РЕПЛІКОВАНИХ БАЗ ДАНИХ

А. В. Горбенко, О. М. Тарасюк

Концепція розподілених реплікованих сховищ даних, таких як Cassandra, HBase, MongoDB була запропонована для ефективного управління великими даними, обсяг яких перевищує можливості традиційних реляційних систем керування базами даних по їх ефективному зберіганню й обробці. Такі системи характеризуються наявністю компромісу між узгодженістю, доступністю, стійкістю до поділу та часовими затримками. Хоча якісні відносини між цими властивостями були раніше визначені в теоремі CAP, проте, актуальною залишається кількісна оцінка ступеня та характеру впливу різних налаштувань узгодженості і тайм-ауту на продуктивність таких систем. У статті представлено результати вимірювання продуктивності Cassandra за допомогою набору тестів YCSB і кількісно показано, якою мірою затримка виконання запитів читання інформації залежить від налаштувань узгодженості то робочого навантаження бази даних. Ці результати ясно показують, що більш висока ступінь узгодженості даних значно збільшує часові затримки, що також узгоджується з якісними висновками теореми CAP. Більш того, показано, що часова затримка та її варіація в значній мірі залежать від поточного робочого навантаження системи. Розподілений характер розглянутих систем не гарантує, що відповідь від бази даних буде отримано протягом встановленого часу очікування. В цьому випадку виникає, так званий часовий збій системи, коли відповідь від неї отримано занадто пізно або ж взагалі не отримано.

У статті аналізується роль тайм-ауту прикладного рівня, який є фундаментальною частиною всіх розподілених механізмів забезпечення відмовостійкості та використовується в якості основного механізму виявлення відмов при роботі в комунікаційному середовищі Інтернет. Зокрема, тайм-аут розглядається в якості основного фактора, що визначає взаємозв'язок між доступністю системи та її швидкодією. Кількісно показано, як різні налаштування тайм-ауту можуть вплинути на доступність системи, а також на середній час обслуговування й очікування обслуговування. Незважаючи на те, що багато сучасних розподілених систем на прикладному рівні використовують статично-заданий тайм-аут, найбільш перспективним підходом є динамічне визначення максимального часу очікування відповіді від системи для забезпечення балансу між продуктивністю та доступністю, а також для підвищення ефективності механізмів відмовостійкості.

Ключові слова: тайм-аут; розподілені бази даних; NoSQL; реплікація; випробування продуктивності; цілісність; доступність; швидкодія; забезпечення компромісу.

ИССЛЕДОВАНИЕ ТАЙМ-АУТА КАК ФАКТОРА ВЛИЯНИЯ НА ПРОИЗВОДИТЕЛЬНОСТЬ И ДОСТУПНОСТЬ РАСПРЕДЕЛЕННЫХ РЕПЛИЦИРОВАННЫХ БАЗ ДАННЫХ

А. В. Горбенко, О. М. Тарасюк

Концепция распределенных реплицированных хранилищ данных, таких как Cassandra, HBase, MongoDB и др. была предложена для эффективного управления большими данными, объем которых превышает возможности традиционных реляционных систем управления реляционными базами данных по их эффективному хранению и обработке. Такие системы характеризуются наличием компромисса между согласованностью, доступностью, устойчивостью к разделению и временными задержками. Хотя качественные отношения между этими свойствами и были ранее определены в теореме CAP, тем не менее, актуальной остается количественная оценка степени и характера влияния различных настроек согласованности и тайм-аута на производительность таких систем. В статье представлены результаты измерения производительности нереляционной базы данных Cassandra с помощью набора тестов YCSB и количественно показано, в какой мере задержка выполнения запросов чтения информации зависит от настроек согласованности и рабочей нагрузки базы данных. Эти результаты ясно показывают, что более высокая согласованность данных увеличивает временные задержки, что согласуется с качественными выводами теоремы CAP. Более того, показано, что временная задержка и ее вариации в значительной степени зависят от рабочей нагрузки системы. Распределенный характер рассматриваемых систем не гарантирует, что ответ от базы данных будет получен в течение конечного времени ожидания. В этом случае возникает, так называемый временной сбой системы, когда ответ от неё получен слишком поздно или же вообще не получен. В статье анализируется роль тайм-аута прикладного уровня, который является фундаментальной частью всех распределенных механизмов обеспечения отказоустойчивости и используется в качестве основного механизма обнаружения ошибок при работе в коммуникационной среде Интернет. В частности, тайм-аут рассматривается в качестве основного фактора, определяющего взаимосвязь между доступностью системы и ее быстродействием. Количественно показано, как различные настройки тайм-аута могут повлиять на доступность системы, а также на среднее время обслуживания и ожидания обслуживания. Несмотря на то, что многие современные распределенные системы на прикладном уровне используют статически-заданный тайм-аут, наиболее перспективным подходом является динамическое определение максимального времени ожидания отклика от системы для обеспечения баланса производительности, доступности и повышения эффективности механизмов отказоустойчивости.

Ключевые слова: тайм-аут; распределенные базы данных; NoSQL; репликация; тестирование производительности; целостность; доступность; быстродействие; обеспечение компромисса.

Горбенко Анатолий Викторович – д-р техн. наук, проф., проф. каф. компьютерных систем, сетей и кибербезопасности, Национальный аэрокосмический университет им. Н. Е. Жуковского «Харьковский авиационный институт», Харьков, Украина; Leeds Beckett University, Leeds, Великобритания.

Тарасюк Ольга Михайловна – канд. техн. наук, доцент, доцент Одесского технологического университета «ШАГ», Одесса, Украина.

Anatoliy Gorbenko – Doctor of Science on Engineering, Professor, Professor of the Department of Computer Systems, Networks and Cybersecurity, National Aerospace University “Kharkiv Aviation Institute”, Kharkiv, Ukraine; Reader with the School of Built Environment, Engineering and Computing, Leeds Beckett University, Leeds, United Kingdom,
e-mail: a.gorbenko@leedsbeckett.ac.uk, ORCID: 0000-0001-6757-1797, Scopus Author ID: 22034015200,
ResearcherID: X-1470-2019, https://scholar.google.com/citations?user=nm8TOtEAAAAJ.

Olga Tarasyuk – PhD, Docent, Associate Professor with the Odessa Technological University STEP, Odessa, Ukraine,
e-mail: O.M.Tarasyuk@gmail.com, ORCID: 0000-0001-5991-8631, Scopus Author ID: 6506732081,
ResearcherID: X-1479-2019, https://scholar.google.com/citations?user=Xmxkp8YAAAAJ.