UDC 004.75.051 **doi: 10.32620/reks.2021.4.13**

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

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

PERFORMANCE EVALUATION OF VARIOUS DEPLOYMENT SCENARIOS OF THE 3-REPLICATED CASSANDRA NOSQL CLUSTER ON AWS

A concept of distributed replicated NoSQL data storages Cassandra-like, HBase, MongoDB has been proposed to effectively manage Big Data set whose volume, velocity and variability are difficult to deal with by using the traditional Relational Database Management Systems. Tradeoffs between consistency, availability, partition tolerance and latency is intrinsic to such systems. Although relations between these properties have been previously identified by the well-known CAP and PACELC theorems in qualitative terms, it is still necessary to quantify how different consistency settings, deployment patterns and other properties affect system performance. This experience report analysis performance of the Cassandra NoSQL database cluster and studies the tradeoff between data consistency guaranties and performance in distributed data storages. The primary focus is on investigating the quantitative interplay between Cassandra response time, throughput and its consistency settings considering different single- and multi-region deployment scenarios. The study uses the YCSB benchmarking framework and reports the results of the read and write performance tests of the three-replicated Cassandra cluster deployed in the Amazon AWS. In this paper, we also put forward a notation which can be used to formally describe distributed deployment of Cassandra cluster and its nodes relative to each other and to a client application. We present quantitative results showing how different consistency settings and deployment patterns affect Cassandra performance under different workloads. In particular, our experiments show that strong consistency costs up to 22 % of performance in case of the centralized Cassandra cluster deployment and can cause a 600 % increase in the read/write requests if Cassandra replicas and its clients are globally distributed across different AWS Regions.

Keywords: *Cassandra; NoSQL; distributed databases; replication; performance benchmarking; YCSB; data consistency; throughput; latency; deployment scenarios; Amazon AWS.*

Introduction

NoSQL (or Not Only SQL) databases are a new generation of distributed data storage that has been recently designed to efficiently deal with rapid data growth [1]. They adhere to the schema-less philosophy and employ horizontal scalability (sharding), Internet-scale replication, and relaxed consistency model to store extremely large datasets and guaranty high throughput, availability and low read/write latency.

NoSQL databases are now widely used in different application domains which generate, store and process BigData. This includes social networks and media, business-critical systems, critical infrastructures, smart industrial applications. For example, Cassandra NoSQL is widely adopted by Uber, Facebook, Instagram, and Netflix. Apple, eBay, GitHub, and the European Organization for Nuclear Research (CERN) use Cassandra either as the main data store or for specific tasks [2].

Attempting to guarantee the atomicity, consistency, isolation, and durability (ACID) of database transactions when storing large distributed datasets results in dramatically increased latency and degraded availability. Thus, NoSQL databases have to sacrifice the ACID concept in favor of the BASE (basically available, soft state, eventually consistent) model [3], which is the price to pay for distributed data handling and horizontal scalability.

The NoSQL ecosystem includes several dozen databases, for instance, Cassandra, HBase, MongoDB, BigTable, Riak, BigTable, Redis, CosmoDB, Neo4J, etc. They cover different application niches by offering various data model categories (e.g. key-value, document, wide-column or graph stores), consistency models, replication strategies and other features [4].

Apache Cassandra is one of the top three in use NoSQL database management systems together with MongoDB and HBase [5]. It is a highly scalable column-oriented database NoSQL database which can store data across many commodity servers in multiple distributed locations [6]. It has a ring-type architecture where data is sharded across all nodes like a logical ring. Cassandra is 'master-less' data storage with no single point of failure, meaning that all nodes are the same and any can receive and process read/write requests. It offers linear scalability, tuneable consistency model and data replication to guaranty high availability and fault-tolerance.

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

Performance evaluation of different NoSQL databases is an active area of research having important practical implication. The primary focus of [7, 8, 9] and other studies is to compare different NoSQL databases based on performance measures. Other works, e.g. [10, 11, 12], use benchmarking results to model and predict databases performance. Despite useful results showing general performance limitations of different distributed data storages existing publications do not examine in details how different factors and settings (deployment pattern, consistency level, replication factor, etc.) affect database latency and throughput. Besides, there has been little efforts (e.g. [13]) made on evaluating scalability and performance of distributed storages considering the impact of distance between nodes or replicas.

Thus, more in-depth analysis studying how different settings and deployment scenarios affect performance of the certain NoSQL database is of a great importance. This work continues a series of related publications evaluating performance of fault-tolerant distributed data storages [14, 15]. It aims at examining the impact of different consistency settings on scalability, latency, throughput of the 3-replicated Cassandra cluster depending on the used single- and multi-region deployment scenario.

1. Cassandra Consistency Model and Deployment Scenarios

1.1. Cassandra tunable consistency model

One of the main features of the Cassandra NoSQL is the tuneable consistency model ranging from weak consistency at one extreme to strong consistency on the other, with varying levels of eventual consistency in between.

It defines a discrete set of consistency settings for every request specifying:

– for READ operations: many replicas that are queried and must respond before the most recent (based on timestamp comparison) read result is returned to the client;

– for WRITE (i.e. INSERT/UPDATE) operations: many replicas that must acknowledge the write operation before it is considered successful (write operations are always sent to all replicas).

The main Cassandra consistency settings include ONE, TWO, THREE, QUORUM, ALL. Additional consistency settings (EACH_QUORUM, LOCAL_QUORUM, LOCAL_ONE) become available if the Cassandra cluster runs across multiple data centres.

Cassandra also employs additional mechanisms to reduce the duration of data inconsistency [6]: hinting, read repair, anti-entropy node repair, NodeSync.

1.2. A notation for describing Cassandra deployment scenarios

The largest unit of Cassandra deployment is a *cluster*. Each cluster consists of nodes from one or more distributed locations. In AWS terms these locations could be composed of separate geographic areas called Regions (e.g. Canada: ca-central-1, Africa: af-south-1, US East: us-east-1, etc.) and/or Availability Zones (isolated locations/datacentres within each Region, e.g. *ca-central-1a*, *ca-central-1b*).

There are currently 25 AWS Regions and 69 Availability Zones (AZ) around the world. Accessibility zones are usually located within 60 miles of each other within a Region and connected with low-latency network links.

In this section we put forward a notation describing distributed deployment of Cassandra cluster and its nodes relatively to each other and to a client (a client could be an application running on the end user devise or some middleware application proxying end-user requests):

– round brackets **()** to define Cassandra cluster;

– curly brackets **{}** to group Cassandra client (C) and nodes (N_i) in the same AWS Region;

– square brackets **[]** to group nodes in the same Availability zone.

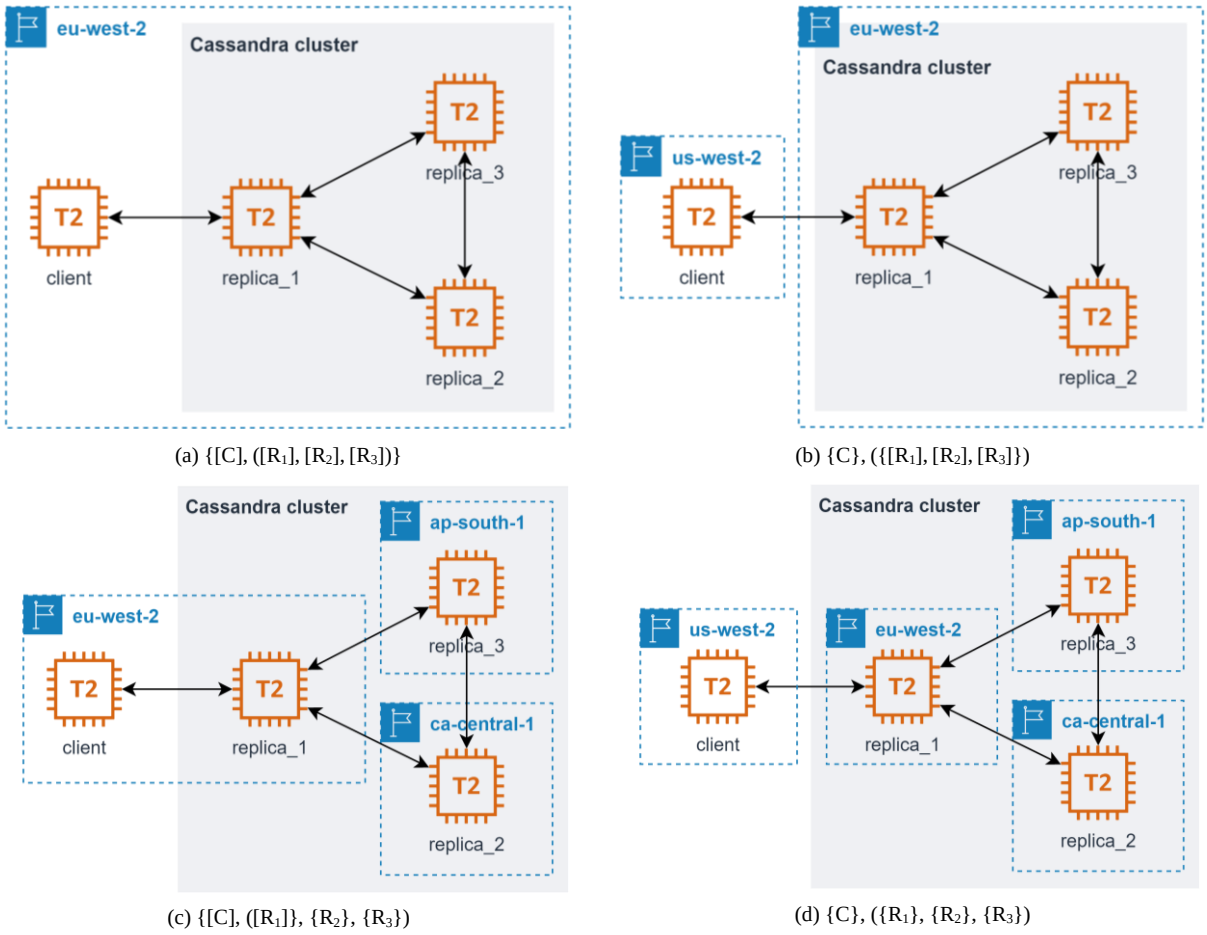
For instance, {[C, (N_1]}, {[N_2], {[N_3]}) deployment record can be read as the three-node Cassandra cluster which nodes are deployed as following: N_1 *node is located together with the client app (*C*) in the same Availability Zone in the same Region; the rest two nodes* N_2 *and* N_3 *are deployed in a different Region, each in a separate Availability zone*.

In case of a completely replicated Cassandra cluster, e.g. when the replication factor is equal to the number of nodes, node symbols (N_i) can be replaced with replica symbols (R_i). For simplicity **{}** or **[]** brackets can be omitted in the deployment record if a client and Cassandra nodes are deployed in the same Region/Availability zone, or when there is only one client/node in a Region.

We consider the following four deployment scenarios of the three-node Cassandra cluster with a replication factor of 3:

a) {[C], ([R_1], [R_2], [R_3])} – a client and all Cassandra replicas are deployed in the same Region, each in a separate AZ;

b) {C}, ({[R_1], [R_2], [R_3]}) – a client (end user application) is located in one geographic region while the Cassandra cluster is in another region with each replica is in a separate Availability zone for better fault-tolerance;



(a) {[C], ([R_1], [R_2], [R_3])}

(b) {C}, ({[R_1], [R_2], [R_3]})

(c) {[C], ([R_1]}, {R_2}, {R_3})

(d) {C}, ({R_1}, {R_2}, {R_3})

Fig. 1. Deployment scenarios of the three-node Cassandra NoSQL cluster with a replication factor of 3

c) {[C], ([R_1]}, {R_2}, {R_3}) – multi-region deployment pattern; a client (a proxy client application/middleware) and one of Cassandra nodes are deployed in the same region, while the rest nodes are globally distributed across the Internet;

d) {C}, ({R_1}, {R_2}, {R_3}) – multi-region deployment pattern; a client (end user application) and Cassandra nodes are deployed across different geographic regions.

A single region deployment pattern is the most common setup for centralized corporate storage systems which data storage nodes and client application(s) generating and consuming data are in the same geographic location. For better fault-tolerance they could be deployed in different AZ within the same region. Multi-region deployment scenarios offer failover and disaster tolerance/recovery. They allow to meet very high availability requirements by deploying nodes/replicas in different geographic regions and can reduce latency by placing data nodes near globally distributed customers. However, requests involving replicas from different regions could be processed much longer due to high inter-region network delay (see Table 1).

Table 1

AWS inter-region latency, ms

AWS Region	us_west_2	ca_central_1	eu_west_2	ap_south_1
us_west_2	2.42	66.71	135.22	221.57
ca_central_1	66.93	3.5	79.87	189.02
eu_west_2	135.56	80.14	3.95	111.69
ap_south_1	221.82	188.88	111.89	3.15

2. Cassandra Performance Benchmarking

2.1. Experimental setup and benchmarking methodology

Performance evaluation methodology used in the paper is similar to one described in [14]. It employs YCSB (Yahoo! Cloud Serving Benchmark) framework widely used to benchmark performance of various relational and non-relational data base management systems [16].

Four Cassandra clusters have been created and deployed on Amazon AWS implementing deployment

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

patterns presented in Fig. 1. Each cluster node was build using the compute-optimized instance type *c3.xlarge* (vCPUs – 4, RAM – 7.5 GB, SSD – 2x40 GB, OS – Ubuntu Server 16.04 LTS).

Unlike other researches analysing performance of distributed data storages (e.g. of [7, 8, 9]) we put the primary focus on analysing Cassandra scalability and examining the impact of data consistency on read/write latency and throughput. For this purpose, the number of threads in our experiments was linearly scaled from 100 to 1000 (until Cassandra performance began to saturate, as it is shown in our previous study [14]). The operation count within each thread was set to 1000.

The above scenario was repeated for consistency settings ONE (the weakest consistency), QUORUM, and ALL (the strongest consistency).

2.2. Cassandra read/write throughput

Fig. 2 and 3 show Cassandra read/write throughput for different deployment scenarios and consistency settings. For example, when a client and all Cassandra nodes are deployed in the same AWS Region is saturated with around 800 threads on average. When Cassandra operates close to its maximal throughput. delays become highly volatile and begin to increase in exponential progression. The presented graphs clearly show that the stronger consistency setting, the lower the throughput. Moreover, the throughput drops dramatically when QUORUM and ALL consistency settings are applied in multi-region deployment scenarios (see Fig. 1, c and Fig. 1, d).

It is also worth noting that we were not able to saturate Cassandra cluster even with 1000 threads in case of (b: all consistency settings) and (d: ONE) deployment scenarios. Thus, Cassandra cluster were not able to achieve its maximum throughput due to overwhelming contribution of the network delay into the overall response time (see Table 1). In all other scenarios the highest Cassandra throughput achieved at peak workload (see Table 2) was close to its maximum/asymptotic throughput.

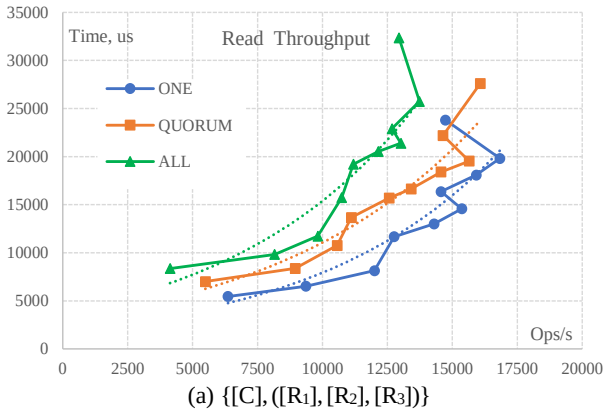
Table 2

The highest Cassandra throughput, ops/s

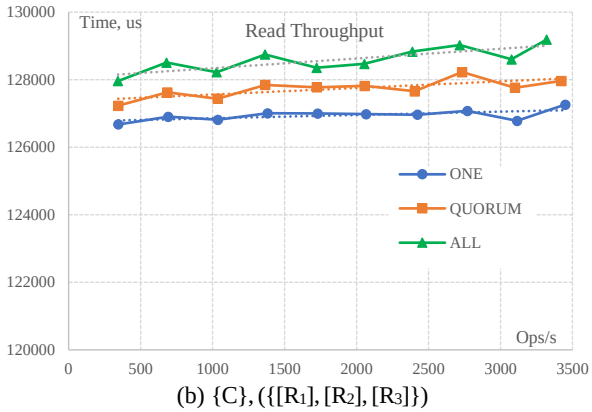
Deployment scenarios	ONE		QUORUM		ALL	
	Read	Write	Read	Write	Read	Write
a: {[C], ([R1], [R2], [R3])}	16830	18338	16074	17384	13735	15434
b: {C}, ({[R1], [R2], [R3]})	3450*	3357*	3424*	3340*	3320*	3319*
c: {[C], ([R1]}, {R2}, {R3})	15160	15906	517	478	355	312
d: {C}, ({R1}, {R2}, {R3})	3381*	3313*	518	491	390	357

* The maximum (asymptotic) throughput was not achieved due to significant network delays

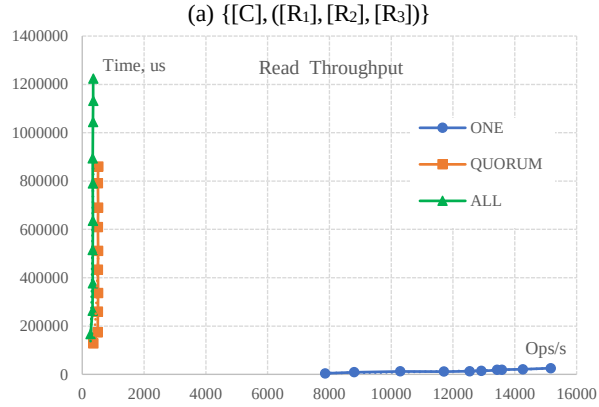
For the single-region Cassandra deployment (a: all consistency settings) and (b: ONE) write throughput overperforms read performance by 9 % on average. This confirm the claim that Cassandra was specially designed as a distributed storage system capable of very high write throughput.



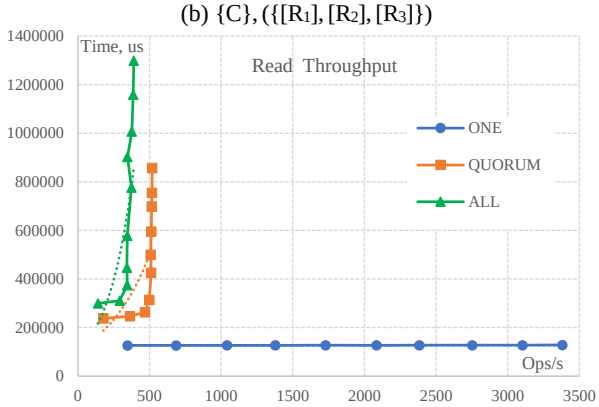
(a) {[C], ([R1], [R2], [R3])}



(b) {C}, ({[R1], [R2], [R3]})



(c) {[C], ([R1]}, {R2}, {R3})



(d) {C}, ({R1}, {R2}, {R3})

Fig. 2. Cassandra READ Trhoughput

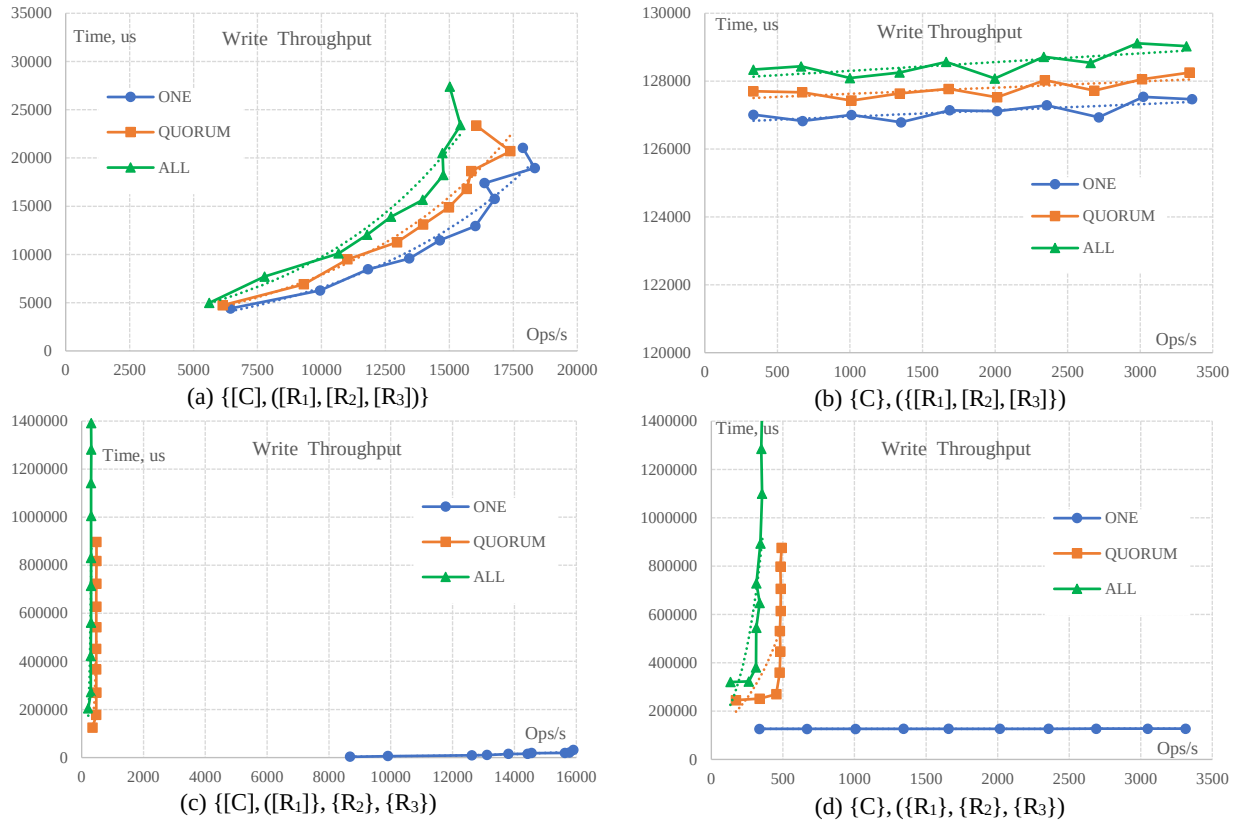


Fig. 3. Cassandra WRITE Trhoughput

Another interesting observation is the fact that read/write throughput in the deployment scenario (a: ONE) overperforms (c: ONE) by 13 % despite the apparent similarity.

This can be explained by the fact that in the case of (c: ONE) deployment all client requests are always sent to the same nearest coordination node in accordance to Cassandra's load balancing policy which takes 'network distance' into account. In scenario (a: ONE) the client workload is equally distributed among all replicas in the same region, which increases the overall throughput.

2.3. Cassandra read/write latency

Cassandra read/write latency statistics is summarized in Tables 3-4. It is shown that the average delay for both read and write requests increases almost linearly as the number of threads increases apart from (d: ONE) and (b: all consistency settings) deployment scenarios for which response latency is almost flat independently on the number of threads. This is due to significant contribution of network delay into the overall response time and inability of a single YCSB client to saturate Cassandra cluster over the Internet.

For the single-region Cassandra cluster deployment (a) latency of read/write performed under the strongest consistency level ALL is higher (by 36 % and 22 % respectively) than the average response time of the weakest consistency setting ONE. For the multi-region deployment (b) these values are 47 and 53 times higher (!) while (b: ONE) deployment is almost as quick as (a: ONE).

When a client and all Cassandra replicas are globally distributed across the Internet (Fig. 1, d), AWS inter-region network delay is the main contributor to read/write latency performed under the ONE consistency setting independently of a number of threads. However, scenario (d: ALL) latency is higher than scenario (d: ONE) latency by an average of 560 % for reads and 600 % for writes.

Because Internet downlinks are generally faster than uplinks in all deployment scenarios except for (a) and partly (c: ONE) write operations were performed slightly longer that reads even despite higher write throughput of the Cassandra NoSQL database.

Conclusions and Lessons Learnt

Availability, consistency and performance of distributed database systems are tightly connected. Although these relations have been identified by the CAP and PACELC theorems in qualitative terms [17, 18], it is still necessary to quantify how different consistency settings, database architectures and deployment scenarios affect system performance and user experience.

In the paper we report results of Cassandra performance benchmarking and examine the impact of different consistency settings on scalability, latency and

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

Table 3

Cassandra READ latency (ms)

	ONE				QUORUM				ALL			
Threads	a: {[C], ([R1], [R2], [R3])}	b: {C}, ({[R1], [R2], [R3]})	c: {{[C], ([R1]}, {R2}, {R3})	d: {C}, ({R1}, {R2}, {R3})	a: {[C], ([R1], [R2], [R3])}	b: {C}, ({[R1], [R2], [R3]})	c: {{[C], ([R1]}, {R2}, {R3})	d: {C}, ({R1}, {R2}, {R3})	a: {[C], ([R1], [R2], [R3])}	b: {C}, ({[R1], [R2], [R3]})	c: {{[C], ([R1]}, {R2}, {R3})	d: {C}, ({R1}, {R2}, {R3})
100	5.45	126.68	6.01	126.41	7.00	127.23	128.13	237.82	8.37	127.96	166.09	299.33
200	6.52	126.90	8.87	126.50	8.37	127.62	174.07	246.77	9.83	128.51	263.51	310.45
300	8.14	126.81	12.33	126.77	10.76	127.44	259.84	262.58	11.72	128.22	376.88	374.32
400	11.67	127.00	11.71	126.74	13.67	127.84	336.05	314.23	15.74	128.75	514.46	446.18
500	13.00	127.00	12.93	126.97	15.70	127.77	432.97	425.84	19.21	128.35	635.51	578.24
600	14.57	126.98	14.32	126.87	16.64	127.81	510.57	500.34	20.53	128.47	790.23	775.99
700	16.36	126.96	18.50	127.31	18.40	127.66	609.47	595.30	21.39	128.83	893.57	902.33
800	18.08	127.08	19.15	127.03	19.54	128.23	689.63	698.66	22.91	129.02	1043.84	1007.11
900	19.81	126.78	20.99	127.38	22.19	127.76	791.19	754.92	25.74	128.60	1131.83	1159.05
1000	23.80	127.26	25.54	128.03	27.59	127.96	859.50	857.24	32.34	129.19	1224.20	1299.01

Table 4

Cassandra WRITE latency (ms)

	ONE				QUORUM				ALL			
Threads	a: {[C], ([R1], [R2], [R3])}	b: {C}, ({[R1], [R2], [R3]})	c: {{[C], ([R1]}, {R2}, {R3})	d: {C}, ({R1}, {R2}, {R3})	a: {[C], ([R1], [R2], [R3])}	b: {C}, ({[R1], [R2], [R3]})	c: {{[C], ([R1]}, {R2}, {R3})	d: {C}, ({R1}, {R2}, {R3})	a: {[C], ([R1], [R2], [R3])}	b: {C}, ({[R1], [R2], [R3]})	c: {{[C], ([R1]}, {R2}, {R3})	d: {C}, ({R1}, {R2}, {R3})
100	4.40	127.01	3.99	126.65	4.73	127.70	123.94	245.55	4.98	128.34	204.11	320.86
200	6.26	126.83	5.68	126.60	6.91	127.67	177.01	252.07	7.71	128.44	271.30	322.85
300	8.46	127.00	9.15	126.91	9.50	127.43	269.54	270.24	10.10	128.09	422.13	381.10
400	9.58	126.78	10.49	126.96	11.25	127.64	366.57	359.27	12.05	128.25	560.22	544.99
500	11.46	127.14	14.82	126.95	13.09	127.77	451.66	447.26	13.91	128.57	713.36	648.06
600	12.94	127.12	15.47	126.97	14.89	127.52	542.01	531.85	15.66	128.08	829.90	728.44
700	15.76	127.29	18.46	127.07	16.80	128.02	627.42	614.31	18.24	128.71	1004.69	893.94
800	17.40	126.93	18.86	127.58	18.64	127.72	722.41	706.08	20.51	128.54	1140.98	1099.53
900	18.96	127.54	20.51	127.67	20.70	128.05	816.88	798.30	23.40	129.12	1281.37	1285.33
1000	21.05	127.47	31.07	127.67	23.36	128.25	896.61	875.72	27.38	129.03	1390.68	1492.45

throughput of the 3-replicated Cassandra cluster depending on the used deployment scenario.

Our experiments confirm a general expectation that stronger data consistency guaranties reduce database throughput and increase latency of read/write operations. However, the single datacentre/region Cassandra deployment offers the best performance for all consistency settings. At the same time, Cassandra can hardly achieve the maximum throughput if its clients are located in other geographic regions. Deployment of a middleware application in the same region as a Cassandra cluster that aggregates and proxies read/write requests from numerous distributed clients can mitigate the dominant impact of high network delays. Another solution which can improve performance of the deployment scenario (b) is implementing asynchronous database requests instead of synchronous ones which block the client until the current operation completes.

Distributing Cassandra nodes across geographic zones close to the database clients also helps to reduce database latency in case of weak consistency settings (ONE, LOCAL_ONE or LOCAL_QUORUM). However, strengthening data consistency by querying replicas from other geographic regions dramatically degrades Cassandra performance and can cause timeout exceptions.

It is worth to remember that timeout settings play an important role of major failure detection mechanism in distributed computer systems [19] and affect efficiency of many Cassandra mechanisms (e.g. speculative retries, hinting, read repairs). Our previous experiments [20] show that the optimal timeout settings should be application specific and need to be adjusted dynamically at run-time taking into account current system workload, consistency settings, deployment scenario and other factors. Setting timeouts dynamically at runtime

can help effectively balance performance, availability, and fault-tolerance of distributed data storages.

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. Apache Cassandra: distributed management of large databases [Electronic resource]. – IONOS. – Access: https://www.ionos.co.uk/digitalguide/hosting/technical-matters/apache-cassandra/ – 10.10.2021.

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

4. Kumar, M. S. Comparison of NoSQL Database and Traditional Database-An emphatic analysis [Text] / M. S. Kumar, P. Jayagopal // Int. Journal on Informatics and Visualization. – 2018. – Vol. 2, No 2. – P. 51-55.

5. 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 – 10.10.2021.

6. Carpenter, J. Cassandra - The Definitive Guide: Distributed Data at Web Scale [Text] / J. Carpenter, E. Hewitt. – O'Reilly Media, 2020. – 400 p.

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. Benchmarking big data systems: A survey [Text] / F. Bajaber, S. Sakr, O. Batarfi, A. Altalhi, A. Barnawi // Computer Communications. – 2020. – Vol. 149. – P. 241-251.

10. 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.

11. 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.

12. 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.

13. Mansouri, Y. The Impact of Distance on Performance and Scalability of Distributed Database Systems in Hybrid Clouds [Text] / Y. Mansouri, M. Ali Babar // ArXiv. – 2020. – Vol. arXiv/2007.15826. – P. 1-26.

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

15. 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.

16. 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.

17. 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.

18. Abadi, D. Consistency Tradeoffs in Modern Distributed Database System Design [Text] / D. Abadi // IEEE Computer. – 2012. – Vol. 45, No.2. – P. 37-42.

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

20. Gorbenko, A. Exploring Timeout as a Performance and Availability Factor of Distributed Replicated Database Systems [Text] / A. Gorbenko, O. Tarasyuk // Radioelectronic and Computer Systems. – 2020. – No. 4. – P. 98-105. DOI: 10.32620/reks.2020.4.09.

References (BSI)

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

2. IONOS, *Apache Cassandra: distributed management of large databases*. Available at: https://www.ionos.co.uk/digitalguide/hosting/technical-matters/apache-cassandra/ (accessed 10.10.2021).

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

4. Kumar, M. S., Jayagopal, P. Comparison of NoSQL Database and Traditional Database-An emphatic analysis. *Int. Journal on Informatics and Visualization*, 2018, vol. 2., no. 2, pp. 51-55.

5. 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 10.10.2021).

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

6. Carpenter, J., Hewitt, E. *Cassandra - The Definitive Guide: Distributed Data at Web Scale*, O'Reilly Media, 2020. 400 p.

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. Bajaber, F., Sakr, S., Batarfi, O., Altalhi, A., Barnawi, A. Benchmarking big data systems: A survey. *Computer Communications*, 2020, vol. 149, pp. 241-251.

10. 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.

11. 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.

12. 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.

13. Mansouri, Y., Babar M. A. The Impact of Distance on Performance and Scalability of Distributed Database Systems in Hybrid Clouds. *ArXiv*, 2020, Vol. arXiv:2007.15826, pp. 1-26.

14. Gorbenko, A., Romanovsky, A., Tarasyuk, O. Interplaying Cassandra NoSQL consistency and performance: A benchmarking approach. *Communications in Computer and Information Science.* Berlin, Springer Nature, 2020, vol. 1279, pp. 168-184.

15. 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.

16. 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.

17. 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.

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

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

20. Gorbenko, A., Tarasyuk, O. Exploring Timeout as a Performance and Availability Factor of Distributed Replicated Database Systems. *Radioelectronic and Computer Systems*, 2020, no. 4, pp. 98-105. DOI: 10.32620/reks.2020.4.09.

Поступила в редакцию 07.11.2021, рассмотрена на редколлегии 26.11.2021

ДОСЛІДЖЕННЯ ПРОДУКТИВНОСТІ РІЗНИХ СЦЕНАРІЇВ РОЗГОРТАННЯ КЛАСТЕРА CASSANDRA NOSQL З ТРЬОМА РЕПЛІКАМИ У ХМАРНОМУ СЕРЕДОВИЩІ AWS

А. В. Горбенко, А. С. Карпенко, О. М. Тарасюк

Концепція розподілених реплікованих нереляційних 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 теоремах CAP та PACELC, проте, актуальною залишається кількісна оцінка ступеня та характеру впливу різних налаштувань узгодженості даних, сценаріїв розгортання та інших властивостей на продуктивність таких систем. У статті аналізується продуктивність кластера даних Cassandra NoSQL та досліджується компроміс між гарантіями узгодженості інформації та продуктивністю в розподілених сховищах даних. Основна увага зосереджена на дослідженні кількісного взаємозв'язку між часом обслуговування Cassandra, її пропускною здатністю та налаштуваннями узгодженості з урахуванням різних сценаріїв розгортання кластеру в одному та кількох хмарних регіонах. У статті наведено результати виконання тестів продуктивності кластера Cassandra з трьома репліками розгорнутого у хмарному середовищі Amazon AWS, що отримані за допомогою набору тестів YCSB. Також авторами запропоновано нотацію для формального опису сценаріїв розподіленого розгортання кластера Cassandra та його вузлів відносно один одного та клієнтів бази даних. Представлено кількісні результати, які показують, як різні налаштування узгодженості та сценарії розгортання впливають на продуктивність Cassandra для різних робочих навантажень. Зокрема, наші експерименти де-

монструють, що строга узгодженість даних коштує в середньому до 22 % продуктивності у разі централізованого розгортання кластера, а також може призвести до збільшення часу виконання операцій читання/запису до 600 % у разі, якщо репліки бази даних та її клієнти глобально розподілені між різними регіонами AWS.

Ключові слова: Cassandra; NoSQL; розподілені бази даних; реплікація; випробування продуктивності; YCSB; узгодженість даних; пропускна здатність; затримка обслуговування; сценарії розгортання; Amazon AWS.

ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ РАЗЛИЧНЫХ СЦЕНАРИЕВ РАЗВЕРТЫВАНИЯ КЛАСТЕРА CASSANDRA NOSQL С ТРЕМЯ РЕПЛИКАМИ В ОБЛАЧНОЙ СРЕДЕ AWS

А. В. Горбенко, А. С. Карпенко, О. М. Тарасюк

Концепция распределенных реплицированных нереляционных 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 теоремах CAP и PACELC, тем не менее, актуальной остается количественная оценка степени и характера влияния различных настроек согласованности данных, паттернов развертывания и других характеристик на производительность таких систем. В статье анализируется производительность кластера данных Cassandra NoSQL и исследуется компромисс между гарантиями согласованности информации и производительностью распределенных хранилищ данных. Основное внимание уделено исследованию количественной взаимосвязи между временем обслуживания Cassandra, ее пропускной способностью и настройками согласованности с учетом различных сценариев развертывания кластера в одном и нескольких облачных регионах. В статье приведены результаты измерения производительности кластера Cassandra с тремя репликами развернутого в облачной среде Amazon AWS, полученные с помощью набора тестов YCSB. Кроме того, авторами предложена нотация для формального описания сценариев развертывания кластера Cassandra и его узлов относительно друг друга и клиентов базы данных. Представлены количественные результаты, которые показывают, как разные настройки согласованности и сценарии развертывания влияют на производительность Cassandra для различных рабочих нагрузок. В частности, наши эксперименты демонстрируют тот факт, что строгая согласованность данных ухудшает производительность кластера в среднем на 22 % в случае его централизованного развертывания, а также приводит к увеличению времени выполнения операций чтения/записи до 600 % в случае, когда реплики базы данных Cassandra и её клиенты глобально распределены между разными регионами AWS.

Ключевые слова: Cassandra; NoSQL; распределенные базы данных; репликация; тестирование производительности; YCSB; согласованность данных; пропускная способность; задержка обслуживания; сценарии развертывания; Amazon AWS.

Горбенко Анатолій Вікторович – д-р техн. наук, проф., проф. кав. комп'ютерних систем, мереж та кібербезпеки, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна; Університет Лідс Бекетт, Лідс, Великобританія.

Карпенко Андрій Сергійович – асп. каф. комп'ютерних систем, мереж та кібербезпеки, Національний аерокосмічний університет ім. М. Є. Жуковського «Харківський авіаційний інститут», Харків, Україна.

Тарасюк Ольга Михайлівна – канд. техн. наук, доцент, доцент Одеського технологічного університету «ШАГ», Одеса, Україна.

Anatoliy Gorbenko – Doctor of Science on Engineering, Professor; 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.

Andrii Karpenko – Ph.D. student with the Department of Computer Systems, Networks and Cybersecurity, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine,
e-mail: a.karpenko@csn.khai.edu, ORCID: 0000-0003-2789-1168.

Olga Tarasyuk – PhD, Docent, Associate Professor with the Odesa Technological University STEP, Odesa, 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.