Performance Measurement of Distributed Storage on Edge Devices

Priyam Loganathan[§]
BITS Pilani, Goa
f20190108@goa.bits-pilani.ac.in

Dhruv Rauthan[§]
BITS Pilani, Goa
f20190095@goa.bits-pilani.ac.in

Animesh Trivedi
VU Amsterdam
a.trivedi@vu.nl

Vinayak Naik
BITS Pilani, Goa
vinayak@goa.bits-pilani.ac.in

2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS) | 978-1-6654-7706-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/COMSNETS56262.2023.10041319

***Abstract*—Edge computing and storage have been proposed as an alternative to current cloud computing infrastructures to address the latter's limitations. However, edge computing services and support for applications based on edge infrastructures are still in the incipient stages. There are numerous storage services for efficient data storage in the cloud. In contrast, few such services exist at the edge. We look at the performance of a cloud storage service Cassandra when deployed on an edge network.**

In this paper, we analyze Cassandra's performance in three different networks. We vary Cassandra's attributes and the overall conditions in each network to simulate a real-world scenario and study the effect on the total time taken for a particular request. We (a) establish trends for different configurations, (b) discuss the reasoning behind them, and (c) suggest optimal configurations for the networks used, which are important for deploying Cassandra in an edge setting.

***Index Terms*—Edge Computing, Cloud Storage, and Performance Analysis**

1 Introduction

In recent years, we have seen a rise in data moving from datacenters to the network edge, inculcating a new paradigm called edge computing [1]. It is a distributed information technology architecture in which client data is processed at the network's periphery, as close to the originating source as possible. Rather than transmitting raw data to a central datacenter for processing and analysis, that work is performed where the data is generated – whether that's a retail store, a factory floor, a sprawling utility, or a smart city. Only the result of that computing work at the edge, such as real-time business insights, equipment maintenance predictions or other actionable answers, is sent back to the primary datacenter in the cloud.

Many applications, such as real-time video processing, augmented/virtual reality gaming, and environment sensing, benefit from such decentralized, close-to-user deployments where low latency and real-time results are expected [2] [3]. With the rise of the edge, the research community has started questioning the general applicability of emerging software and services designed for cloud computing, such as augmented reality and industrial IoT. The primary motivating bottleneck is the rather long end-to-end cloud access latency due to the limited and sparse deployment of datacenters across the globe [4].

Charyyev et al. [4] evaluated the current state of cloud connectivity globally and concluded that most of the world's population could access a cloud facility within 100ms. Low latency devices can function efficiently only in continents such as the US and Europe, where cloud provider datacenters are abundant. Extensive datacenter deployment is necessary to make cloud access latencies consistently compatible with the requirements of next-generation applications, especially for Asia, South America, and Africa. An alternative to this expensive solution of constantly building new datacenters, is edge computing, where full-fledged datacenters are not mandatory to ensure low latency.

To draw parallels to large cloud datacenters, we chose NoSQL, which is used for managing and storing large databases. Apache Cassandra is an open-source, distributed NoSQL database used for handling large amounts of data across multiple servers [5]. It provides high availability without a single centralized point of failure since it consists of a network with hundreds of nodes. One key component of Cassandra's architecture is replication. Using replication, each data item is replicated at N hosts, where N is the replication factor. This scheme allows datacenters to handle certain node failures without any particular outage affecting the user [5]. We chose to use Cassandra due to its highly distributed nature, similar to an edge network that focuses on a decentralized architecture. The general network architecture for an edge Cassandra network is illustrated in Figure 1. We will not be using the cloud network as we want to focus on only the edge network's performance for distributed storage systems.

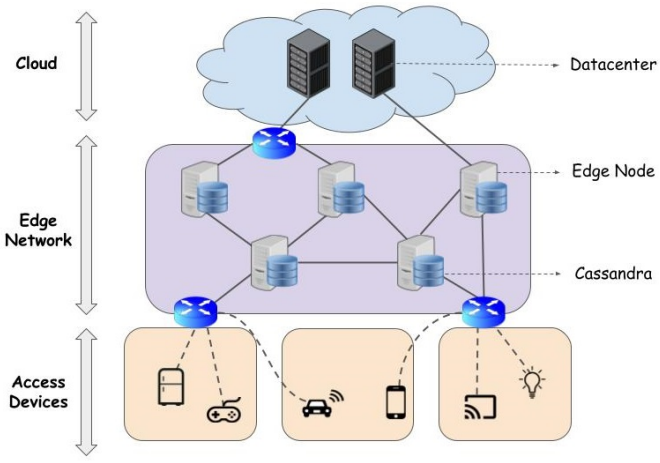

Figure 1: Edge Cassandra Architecture

[§]Equal contribution

We aim to analyze the unique challenges and issues posed by edge networking to cloud-based storage services. In our work, we start with understanding and quantifying the impact edge networking has on the workings of Cassandra by evaluating its performance. We present data and analyze it by comparing inputs to Cassandra parameters, such as replication.

2 Related Work

Database solutions for the edge have been primarily focused on satisfying the needs of IoT infrastructure. Billions of deployed IoT devices generate zettabytes (ZB) of data [6] [7]. This proliferation of data, mainly consisting of time series, poses new challenges for data collection, processing, storage, and analysis. In the cloud, we have more than half a dozen key-value storage, file systems, and database services to efficiently store data and the runtime state of applications. However, no such service exists at the edge yet .

Although existing databases can process large-scale time-series data streams on cloud-based infrastructure, they are not well designed to run on edge nodes with limited hardware resources, power budget, and scalability [8]. EdgeDB and VergeDB are two database solutions that were proposed to serve these needs. VergeDB is an edge-based system that can rapidly ingest data from sensors while optimizing for compression, aggregation, and filtering based on the needs of a downstream analytics consumer [9]. EdgeDB is a time series database that can store and query massive time-series data streams on edge nodes. The database is desired to have several necessary capabilities, including high insertion throughput, high write performance, low query response time, and low resource requirement. Other research projects have targeted the edge's storage, such as PathStore, a database for a path computing platform [10] and C-SPOT, which couples FaaS function invocation with a persistent storage [11].

Despite these solutions, a quantitative evaluation of storage concerns at the edge is still missing. Before building another database for the edge, we decided to observe how a popular cloud database such as Cassandra would perform on edge. We achieve this by varying network performance characteristics and substituting different Cassandra attributes such as replication factor to determine whether an entirely new storage system is actually needed.

Evaluation and benchmarking of different cloud storage systems has been a source of inspiration for our work. There have been several papers that contain NoSQL database comparison and performance analysis. Cassandra has been benchmarked using Yahoo! Cloud Serving Benchmark (YCSB) [12], which is often used to compare the relative performance of NoSQL database management systems, by Barata and Bernardino [13] on one and three-node clusters. An extensive evaluation of a cloud storage system on an edge network is still missing. Further, it has also been evaluated using different data partitioning strategies, such as RandomPartitioner and ByteOrderedPartitioner, but for a single consistency level of one for the experiments [14]. Comparison of Cassandra against other databases, such as MongoDB, also uses static values for replication factor and consistency level across their experiments [15].

Furthermore, the article [16] assesses Cassandra's scalability; considering the factors of workload, data size, and the number of simultaneous sessions, they increase only the number of nodes. The study shows that increasing only the number of nodes does not guarantee performance improvement and that Cassandra manages the simultaneous requests of threads well.

Hence, it is important to understand and evaluate Cassandra's performance for different configurations within the same node setup to optimize it for particular networks.

3 Setup of Emulation

3.1 Cassandra

We chose Cassandra as our storage solution due to a number of reasons outlined below. Most importantly, it is not dependent on a single centralized server, i.e, it is decentralized and fulfills the needs of an edge network. It also provides strategies that allow the data to be nearer to the users, one of the core concepts involved in edge storage systems.

The smallest component of Cassandra is called a $node$, where the object data is stored. A $datacenter$ is a group of nodes configured within a cluster for replication. And a $cluster$ is the outermost storage container that contains one or more datacenters. Cassandra uses a gossip protocol to keep data consistent between all the different nodes in the cluster. It is a peer-to-peer protocol that runs periodically every second and exchanges messages with up to three nodes in the cluster. This way, every node knows where an object is without sending extra messages requesting a particular object's location.

Furthermore, replicas of each object are stored on multiple nodes to ensure reliability and fault tolerance. The total number of replicas stored is referred to as the Replication Factor (RF). We vary this parameter throughout our experiments. The replication strategies offered include SimpleStrategy and NetworkTopologyStrategy [17]. SimpleStrategy is used for a single datacenter and places the first replica on a node determined by the partitioner. NetworkTopologyStrategy is preferred for a multiple datacenter deployment and it lets us define how many replicas we want in each datacenter. We use SimpleStrategy for our experiments.

Finally, another attribute, Cassandra's Consistency Level (CL) is the minimum number of Cassandra nodes that must acknowledge a read or write operation before it can be considered successful. The CL is always less than or equal to the RF for a particular configuration. It provides a trade-off between object access times and its reliability. For a high CL, more acknowledgments are required for a particular request, increasing the total access time. In contrast, for a low CL, the data is stored at fewer nodes, compromising reliability.

3.2 Continuum

Continuum is a deployment and benchmarking framework for the edge continuum developed at VU Amsterdam [18]. It automates setting up and configuring emulated cloud, edge, and endpoint hardware and networks. Further, it manages the installation of the software inside the emulated environment and can perform benchmarks as well. The execution consists of three phases – infrastructure, installation, and benchmarking. In our research, we have used it solely for the deployment of edge nodes. It is to be noted that the benchmarking features of the continuum are not being used because it is still under active development. Alternatively, we have written Python scripts to evaluate Cassandra manually. The framework uses the following software.

- **KVM** Kernel-based Virtual Machine (KVM) is an open-source virtualization technology built into Linux. It lets us turn Linux into a hypervisor that allows a host machine to run multiple isolated virtual environments called virtual machines (VMs).
- **Libvirt** This is an open-source management tool for managing platform virtualization. The *virsh* console uses the interface provided by *libvirt* to interact with the VMs.
- **QEMU** This is a free and open-source emulator. It emulates the machine's processor and provides a set of different hardware and device models for the machine, enabling it to run a variety of guest OSes.
- **Ansible** An open-source software that provides a framework used to automate IT operations. This is used to install the required software, such as Docker, Kubernetes, and KubeEdge, on the VMs.

This framework is open-source and available on Github [18].

4 Methodology of Measurement

We consider an edge network containing a cluster of nodes that also run Cassandra as their primary distributed storage system. The configuration consists of k edge nodes (k = 3, 4, or 5) and a single endpoint. All these devices are run on QEMU virtual machines with appropriate changes being made to the Cassandra *.yaml* configuration file. This simulates a scenario wherein a user (the endpoint) writes data to its database, distributed between the k edge nodes. A JSON file that contains the sample testing data objects is parsed and passed as parameters to a Cassandra query. The DataStax Python driver for Cassandra helps create scripts to execute these queries for write operations to the database.

There are four control parameters – latency, packet loss, RF, and the number of nodes in the topology. These will be changed throughout our experiment to test Cassandra's performance. Our code is available on Github [19].

4.1 Latency

We explore the effects of changing latencies between the edge nodes. *tcconfig*, a Linux traffic control command wrapper, adds delays between the different edge nodes and between a node and the endpoint. Network latencies are a good replacement for geographical distances. A user may be at different geographical locations while accessing the edge database. Network impairments such as network congestion, packet losses, and retransmission result in latency fluctuations [20]. In addition, increasing latency also has a considerable effect on the TCP throughput [21]. Understanding the impact of link latency on Cassandra's performance in an edge-based data storage layout is crucial. We increase the values up to 100ms as most of the world's population can access a cloud facility within that time [4].

4.2 Packet loss

Packet loss is another fundamental network metric affecting network application performance. Hence, it is critical to understand the trends and impact of packet loss on distributed applications and storage systems [22]. We use *tcset*, a command of *tcconfig*, to add packet loss to particular interfaces and emulate real-life scenarios. Ideally, 0-1% packet loss is acceptable for the majority of the applications. The quality is significantly affected when the packet loss reaches values above 5%, meaning the network's performance has been highly degraded [23]. Keeping these ranges in mind, we chose the packet loss values as (a) Low = 0.5%, (b) Medium = 2.5%, and (c) High = 7.5%.

4.3 Replication factor

Our third control parameter is RF. Every k-node cluster can have RF values between 1 to k, and we use all possible combinations. Further, CL provides a trade-off between object access times and its reliability. Its default value is defined by the $LOCAL\_QUORUM$, which equals (RF/2) + 1. We will take these values defined by the cluster for each RF.

4.4 Node scalability

Ideally, an application requires Cassandra to consist of more than three nodes for storage. In this experiment, we increase the number of nodes to four and then five and repeat the latency and packet loss experiments. This is done to compare the results from the previous experiment and establish a trend when increasing the nodes in the topology. Extra nodes are deployed using the Continuum framework, and all the VMs in our experiment are equal in performance and storage to maintain uniformity.

By varying the input parameters, we measure the time taken by Cassandra to finish a thousand write requests. The request raised by the endpoint reaches any node in the cluster, which then acts as the coordinator node for that request. According to the nature of the request, it either reads or writes to several nodes specified by the consistency level and the hash function, which determines how data is distributed across the nodes in the cluster. After the coordinator node receives an acknowledgment from all the involved nodes, it forwards the appropriate response to the endpoint. The time taken from sending the request to receiving the response by the endpoint is indicative of Cassandra's performance.

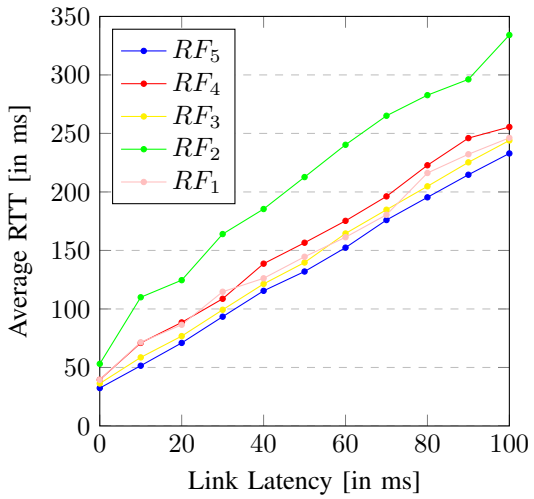

Figure 2: Average performance for a write request in a 5-node cluster with high packet losses. RF_2 shows relatively worse performance.

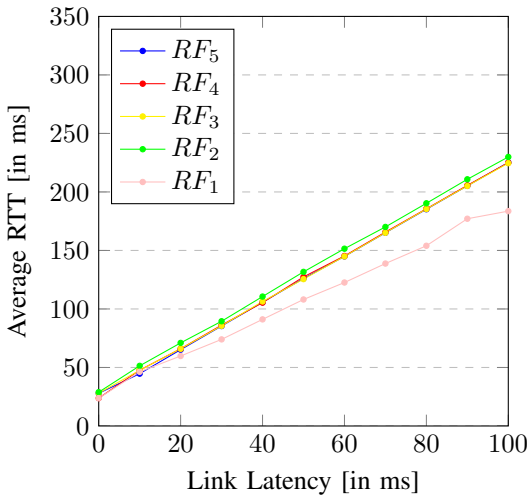

Figure 3: Average performance for a write request in a 5-node cluster with low packet losses. RF_1 exhibits least RTT.

5 Results

We observe several trends while plotting the graphs for different configurations and infer the following.

5.1 RF_2 anomaly

Figure 2 shows the RTT of a write request in a 5-node cluster with high packet loss. The legend denotes the different RFs possible and the latency between each edge node increases along the x-axis. We can infer that an RF of 2 (RF_2) results in the worst performance compared to other replication factors under a 5-node setup and high packet loss. RF_2 performed similarly to others under low (Figure 3) and medium packet losses, in contrast to high packet loss. The difference between RF_2 and the others gradually increases, indicating that RF_2 highly depends on packet loss. Formulating a median line for each replication factor in the graph shows us that the slope of the RF_2 line is more significant than any other slope. Consequently, as the latency between the nodes increase, the performance of Cassandra under RF_2 worsens more quickly than any other replication factor. Similar behavior is observed in the 3 and 4-node clusters. All these observations lead to the conclusion that RF_2 should be used only in networks with low to medium packet loss.

5.2 RF_5 performance

For a high packet loss 5-node cluster, from Figure 2, RF_5 gives the best performance. Interestingly, this suggests that the RF should be increased if a network worsens. This is contrary to the intuitive thinking that in this scenario, increasing the RF would simply lead to more overall requests, more packet drops, and, consequently, worse performance. The performance improves and can be attributed to how Cassandra returns write acknowledgment packets. Hence, the maximum RF for a 5-node cluster is best for bad network conditions. This behavior is explained in detail in Section 6.

5.3 RF_1 performance

Figure 3 shows that setting the cluster configuration to RF_1 gives the best performance in a 5-node cluster with a low packet loss network. The 3-node and 4-node setups under the same network conditions indicate that RF_1 performs best.

However, as mentioned above, using RF_1 compromises data availability, which might result in some data loss. An RF of 1 means a single copy of the data is maintained. So, if the node with the data fails, the information is lost. This is not a reliable configuration in a production environment and should be used only when the user can afford to lose data.

For low packet loss, RF_1 shows the best performance in all the clusters. However, as packet loss increases, RF_1's performance degrades faster than other RFs, eventually matching or surpassing them. We can observe this in Figure 2, where for the high packet loss 5-node cluster, RF_1's slope is similar to RF_3, RF_4, and RF_5. Similar trends are observed in the other clusters as well.

5.4 Increasing number of nodes without sacrificing performance

Taking the standard deviation of RF_3's average RTT for the 3, 4, and 5-node clusters, we found that this value is < 1.5ms for any link latency. Based on this observation, we can conclude that when the Cassandra cluster is in RF_3 configuration, it performs the same across all numbers of nodes irrespective of packet loss. If the cluster is set to RF_3, the database administrator can increase the number of nodes in the cluster and expect the same performance irrespective of the network conditions.

5.5 Optimal replication factor

Taking the median for different packet losses, RF_3, RF_4, and RF_5 show similar performance, as shown in Figure 4. If one is hosting Cassandra on a cloud service such as Google Cloud or AWS, where they are not aware of the exact network

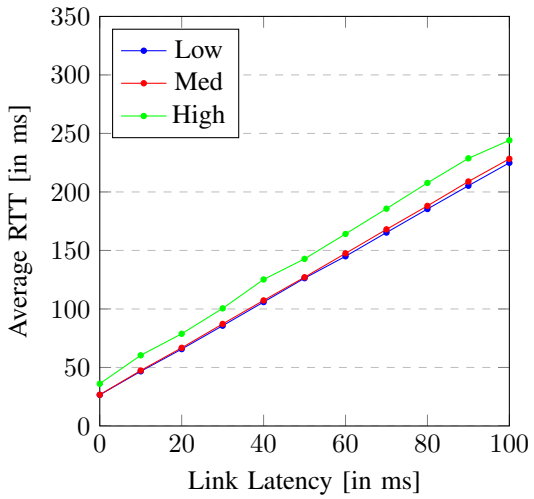

Figure 4: Performance of the average of RF_3, RF_4, and RF_5 in terms of packet losses. RF $>$ 2 show similar performance.

conditions (namely packet loss), for consistent performance, they should prefer RF $>$ 2 for these clusters. RF_1 shows better performance but comes at the cost of only storing data in 1 node, which is especially dangerous if that node fails in the cluster. RF $>$ 1 ensures reliability and fault tolerance.

6 Discussion

The specific behavior of certain RFs can be attributed to how Cassandra distributes and manages requests sent by the endpoint. The DataStax Python Driver uses the RoundRobin-Policy as the default load-balancing policy. This ensures that the endpoint evenly distributes queries across all nodes in the cluster, regardless of what datacenter the nodes may be in. Verifying this with traffic inspection, we can confidently say that, for a 5-node cluster, each node will act as the coordinator 200 times for a total of 1000 write requests. Additionally, Cassandra distributes data across the cluster using a Consistent Hashing algorithm. A partitioner is a function that determines how data is distributed across the nodes in the cluster. It is used to derive a token representing a row from its partition key using the abovementioned algorithm. Each row of data is then distributed across the cluster according to the token's value.

Defining P_{nc} as the probability that one of the nodes containing the data is not the coordinator node, and P_c as the probability that one of the nodes containing is the coordinator node. ρ_{x_total} are the number of acknowledgment requests the coordinator node sends out, and ρ_{x_ack} are the number of acknowledgments that it needs according to the consistency level in the cluster, where x can be nc or c depending on the scenario. We can say

$$\phi = P_{nc}(\rho_{nc\_ack}/\rho_{nc\_total}) + P_c(\rho_{c\_ack}/\rho_{c\_total})$$

where ϕ is the ratio of packets required for the acknowledgment to the total requests sent by the coordinator. The higher that ϕ is, the more impact packet loss has on the total RTT for a single request. We'll take an example from the 5-node cluster as shown in Figure 5.

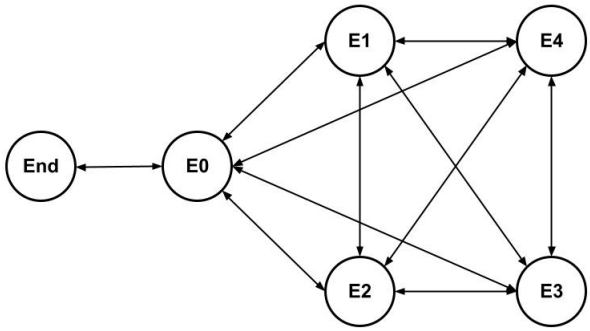

Figure 5: 5-node cluster topology

For RF_2, suppose that the data is to be written in E3 and E4. The probability of any other node becoming the coordinator is 0.6 since 3 out of 5 nodes are left. The coordinator will send two write requests and expect two acknowledgments back, as CL2 is the default for this configuration. If E3 or E4 (probability 0.4) becomes the coordinator, it will only send and expect one acknowledgment back from the other node. The other acknowledgment is not required as the coordinator node can retrieve the data from itself. In this case, $\phi = 0.6(2/2) + 0.4(1/1) = 1$, which is the maximum value possible. ϕ will stay the same not just for E3 and E4, but for any combination of 2 nodes.

In contrast, when the cluster employs RF_4, we can assume the data is to be written in E1, E2, E3, and E4. The probability of any other node (only E0 is possible) becoming the coordinator is 0.2. Even though the CL is 3, E0 will send write requests to all other 4 nodes but will expect acknowledgments back from 3 of them. The other case remains similar, where the coordinator node sends write requests to the other 3 nodes and only expects acknowledgments from two of them. Here, $\phi = 0.2(3/4) + 0.8(2/3) = 0.68$, which is lower than the ϕ for RF_2.

Excluding RF_1, as the coordinator node may in some cases directly respond to the endpoint, in our 3, 4, and 5-node clusters, the ϕ for RF_2 is always one, as shown in Table 1. This explains the unusually high RTT for RF_2 across these clusters. Increasing the packet loss gives the cluster little room for error since every dropped packet results in the coordinator node sending the request for acknowledgment again, increasing the total time for the write request.

RF	3-node	4-node	5-node
2	1	1	1
3	0.5	0.54	0.56
4	-	0.66	0.68
5	-	-	0.5

Table 1: ϕ for different node clusters

7 Conclusion

We analyzed the workings of a cloud storage database on an edge setup under various networking circumstances. There have been many attempts to build a database suitable for the edge however we decided to step back and test an existing popular cloud storage system on a simulated edge setup. We used the Continuum framework to deploy an endpoint node and Cassandra on multiple edge nodes and recorded the average RTT taken by write requests, which gave us information about the database performance in different configurations and network conditions.

Using graphs, we established trends for certain settings and proposed an explanation for the anomalies encountered. We also identified the best-performing configurations for 3, 4, and 5-node clusters and recommended particular configurations according to network conditions. In our future work, we aim to extend and establish a pattern for a cluster containing any number of nodes.

References

[1] Shi, Weisong, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. "Edge computing: Vision and challenges." IEEE internet of things journal 3, no. 5 (2016): 637-646.

[2] Choy, Sharon, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. "A hybrid edge-cloud architecture for reducing on-demand gaming latency." Multimedia systems 20, no. 5 (2014): 503-519.

[3] Grassi, Giulio, Kyle Jamieson, Paramvir Bahl, and Giovanni Pau. "Parkmaster: An in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments." In Proceedings of the Second ACM/IEEE Symposium on Edge Computing, pp. 1-14. 2017.

[4] Charyyev, Batyr, Engin Arslan, and Mehmet Hadi Gunes. "Latency comparison of cloud datacenters and edge servers." GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE, 2020.

[5] Lakshman, Avinash, and Prashant Malik. "Cassandra: a decentralized structured storage system." ACM SIGOPS Operating Systems Review 44, no. 2 (2010): 35-40.

[11] Wolski, Rich, Chandra Krintz, Fatih Bakir, Gareth George, and Wei-Tsung Lin. "Cspot: Portable, multi-scale functions-as-a-service for iot." In Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, pp. 236-249. 2019.

[12] Cooper, Brian F., Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. "Benchmarking cloud serving systems with YCSB." In Proceedings of the 1st ACM symposium on Cloud computing, pp. 143-154. 2010.

[13] Barata, Melyssa, and Jorge Bernardino. "Cassandra's Performance and Scalability Evaluation." In International Conference on Data Management Technologies and Applications, vol. 2, pp. 127-134. SCITEPRESS, 2016.

[6] S. George. IoT Signals report: IoT's promise will be unlocked by addressing skills shortage, complexity and security., 2019 (accessed August 15, 2020). https://blogs.microsoft.com/blog/2019/07/30/.

[7] Hung, Mark. "Leading the iot, gartner insights on how to lead in a connected world." Gartner Research 1 (2017): 1-5.

[8] Yang, Yang, Qiang Cao, and Hong Jiang. "EdgeDB: An efficient time-series database for edge computing." IEEE Access 7 (2019): 142295-142307.

[9] Paparrizos, John, Chunwei Liu, Bruno Barbarioli, Johnny Hwang, Ikraduya Edian, Aaron J. Elmore, Michael J. Franklin, and Sanjay Krishnan. "VergeDB: A Database for IoT Analytics on Edge Devices." In CIDR. 2021.

[10] Mortazavi, Seyed Hossein, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, and Eyal De Lara. "Cloudpath: A multi-tier cloud computing framework." In Proceedings of the Second ACM/IEEE Symposium on Edge Computing, pp. 1-13. 2017.

[14] Dede, Elif, Bedri Sendir, Pinar Kuzlu, Jessica Hartog, and Madhusudhan Govindaraju. "An evaluation of cassandra for hadoop." In 2013 IEEE Sixth International Conference on Cloud Computing, pp. 494-501. IEEE, 2013. Harvard

[15] Araujo, Jose Maria A., Alysson Cristiano E. de Moura, Silvia Laryssa B. da Silva, Maristela Holanda, Edward de Oliveira Ribeiro, and Gladston Luiz da Silva. "Comparative Performance Analysis of NoSQL Cassandra and MongoDB Databases." In 2021 16th Iberian Conference on Information Systems and Technologies (CISTI), pp. 1-6. IEEE, 2021.

[16] Abramova, Veronika, Jorge Bernardino, and Pedro Furtado. "Testing cloud benchmark scalability with cassandra." In 2014 IEEE World Congress on Services, pp. 434-441. IEEE, 2014.

[17] DataStax. "Data replication". DataStax Documentation. November 1, 2022. https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/architecture/archDataDistributeReplication.html

[18] Matthijs Jansen. 2022. Continuum. https://github.com/atlarge-research/continuum (2022)

[19] Dhruv Rauthan. 2022. Performance-Measurement-of-Distributed-Storage-on-Edge-Devices. https://github.com/dhruvrauthan/Performance-Measurement-of-Distributed-Storage-on-Edge-Devices (2022)

[20] Gettys, Jim. "Bufferbloat: Dark buffers in the internet." IEEE Internet Computing 15.3 (2011): 96-96. APA

[21] Padhye, Jitendra, et al. "Modeling TCP throughput: A simple model and its empirical validation." Proceedings of the ACM SIGCOMM'98 conference on Applications, technologies, architectures, and protocols for computer communication. 1998. APA

[22] Wang, Y. Angela, et al. "Queen: Estimating packet loss rate between arbitrary internet hosts." International Conference on Passive and Active Network Measurement. Springer, Berlin, Heidelberg, 2009. APA

[23] Pundir, Meena, and Jasminder Kaur Sandhu. "A systematic review of quality of service in wireless sensor networks using machine learning: Recent trend and future vision." Journal of Network and Computer Applications 188 (2021): 103084.