Distributed Streaming Storage Performance Benchmarking: Pravega and Pulsar

Ramesh Kadaba Vasudevamurthy
Airspan Networks, Visvesvaraya Technological University, Belagavi, India
kvramesh@gmail.com (corresponding author)

G. T. Raju
Department of Computer Science and Engineering, SJC Institute of Technology, Visvesvaraya Technological University, Belagavi, India
gtraju1990@yahoo.com

Received: 10 June 2024 | Revised: 27 June 2024 | Accepted: 5 July 2024

 DOI: https://doi.org/10.48084/etasr.8076

ABSTRACT

Massive data shoving can reach the greatest throughput, which is necessary for distributed streaming storage to function at its best. The comparison of the distributed streaming storage systems Pulsar and Pravega for a given number of producers and data packet size is covered in detail in this study. This analysis' benchmark tool accommodates several producers and consumers. When connection pooling is enabled and 0.5 million records are thrust at a 10 Mbps data rate, both streaming storages are assessed for latency percentile comparison. A novel idea called sbk-charts is introduced in the current study, which can create practical charts from CSV files. Multiple CSV files can be joined by sbk-charts to construct a single combined xlsx file with helpful charts. The outcomes of the experiment are then evaluated for performance comparison in a number of dimensions.

Keywords-benchmarking, throughput; Pravega; Pulsar; storage benchmarking kit; latency; connection pooling

I. INTRODUCTION

Pravega [1, 2] and Pulsar [3] are two distributed streaming platforms that offer dependable real-time data processing and storage. While their architectures, functionalities, and use cases are somehow similar, they differ significantly. Choosing between them depends on certain factors, including the necessity for robust ordering, preferred data models, integration requirements, and the platform ecosystem [4]. This study conducts a thorough set of tests to determine the performance comparison between Pulsar and Pravega. In all scenarios, SBK is utilized to push data to the two storage systems under test.

II. STORAGE BENCHMARK KIT

An open-source software framework for measuring storage performance and providing real-time analytics on any type of storage system is the Storage Benchmark Kit (SBK) [5]. It is written in Java with some of its utilities developed in Python. Popular storage systems can be measured for performance using SBK. Any payload, including strings, byte arrays, and byte buffers, can be used with SBK. SBK does performance measurements with precision in ms, μs, and ns deploying any time stamp. Throughput, different latency percentiles, and Grafana [6] graphs are produced by SBK to facilitate simple visual analysis of performance statistics. Given that SBK is very scalable in terms of CPU and memory, greater system resources on an SBK server will result in higher performance metrics. SBK also provides a framework to add any new storage systems. The Gradle command on SBK creates a template where the developer must fill in the read/write Java APIs for the new storage system. The SBK application has the below variants:

- SBK-YAL (Argument Loader): It takes the yml file as argument which will have all the arguments fed in.
- SBK-RAM (Results Aggregation Monitor): It runs in a GRPC server, collects the performance results from multiple SBK instances, and produces consolidated throughput and latency values along with graphs.
- SBK-GEM (Group Execution Monitor): It combines SBK RAM and SBK. In the former the SBK instances can be just executed on multiple hosts with a single SBK-GEM command.

The SBK operates in 4 different modes.

- Burst Mode: In this mode, SBK pushes/pulls the messages to/from the storage client (device/driver) as much as possible. This mode is used to find the maximum throughput that can be obtained from the storage device or storage cluster (server). This mode can be utilized for both

writers and readers. By default, the SBK runs in the Burst Mode.

TABLE I. PULSAR - PRAVEGA COMPARISON

	Pravega	Pulsar
Architecture	Built on top of Apache Bookkeeper. Uses streams which are unbounded sequences of events that can be written to and read from. Supports high write and read throughput.	Designed on concepts of topics and subscriptions. Uses Apache Bookkeeper [7] for storage and Apache Zookeeper [8, 9] for coordination. Supports multi-tenancy to manage workload on same cluster.
Data model	Provides a byte-oriented storage model, where data are written and read as a sequence of bytes. Supports arbitrary payload formats and gives users full control over the data representation	Supports both byte-oriented and message-oriented storage models. In the message-oriented model, data are encapsulated in messages with metadata, making it easier to work with structured data and enforce schema validation.
Durability and consistency	Offers strong durability guarantees by persisting data to multiple BookKeeper nodes. It ensures that data are reliably stored and available even in the event of failures. Supports exactly once processing semantics, which provides consistency guarantees for data consumption	Provides similar durability guarantees by storing data in the Apache BookKeeper, which replicates data across multiple nodes. It supports at-least-once message delivery semantics by default, but it also provides mechanisms for achieving exactly once semantics through deduplication
Ecosystem and integration	Relatively smaller ecosystem. Provides client libraries in multiple programming languages and supports integration with Apache Flink [10], Apache Samza [11], and other stream processing frameworks	Vibrant and growing ecosystem. Offers a wide range of client libraries, including Java and Python, C++. Pulsar integrates well with popular stream processing frameworks like Apache Flink, Apache Beam, and Apache Spark.
Use cases	Suitable for scenarios that require precise ordering of events and strong durability guarantees. It is well-suited for event sourcing, real-time analytics, and applications that require high-throughput data ingestion and processing	Suitable for various use cases, including real-time messaging, event-driven architectures, IoT data ingestion, and microservices communication

- Throughput Mode: In this mode, the SBK pushes/pulls/from the messages to the storage client (device/driver) with specified approximate maximum throughput in terms of MB/s. This mode is implemented to find the least latency that can be obtained from the storage device or storage cluster (server) for given throughput.
- Rate Limiter Mode: This mode is another form of controlling writers/readers throughput by limiting the number of records per second. In this mode, the SBK pushes/pulls the messages to/from the storage client (device/driver) with specified approximate maximum records per sec. This mode is used to find the least latency that can be obtained from the storage device or storage cluster (server) for events rate.
- End to End Latency Mode: In this mode, the SBK writes and reads the messages to the storage client (device/driver) and records the end-to-end latency. End-to-end latency means the time duration between the beginning of the writing event/record to stream, and the time after reading the event/record. In this mode, the user must specify both the number of writers and readers. The -throughput (Throughput Mode) or -records (late limiter) options can be used to limit the writer's throughput or records rate.

III. COMPARISON SETUP DETAILS

Pravega container clusters were run on an Ubuntu server having 4 cores and 32 GB RAM with 2 TB Disk space storage. Once the containers are instantiated, the cluster status can be checked with: *docker-compose up –d*. Pulsar docker was run on the same Ubuntu server. It was made sure that Pravega docker containers are terminated when the Pulsar docker standalone container is running. This way it was certified that the host environments for Pulsar and Pravega were the same.

TABLE II. SYSTEM COMPORENTS AND VERSIONS

Component	Remarks
SBK	Version 5.1
Pravega	Version 0.13.0
Pulsar	Version 3.1
Zookeeper	Version 3.6.1
Bookkeeper	Version 4.16.3
Grafana	version 9.0.6
Host server: Pravega/Pulsar	4 cores and 32 GB RAM, 2 TB disk

Running Pulsar docker: *docker run -it -p 6650:6650 -p 8080:8080 apachepulsar/pulsar:latest bin/pulsar standalone*

Running Pravega docker-compose result can be seen in Figure 1.

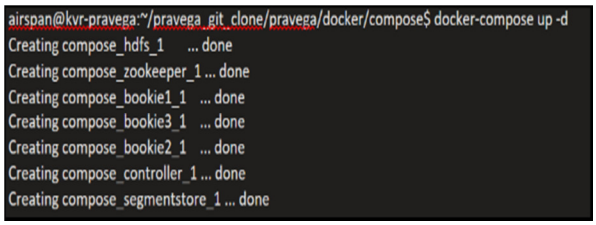


Fig. 1. Running Pravega docker-compose result.

SBK was running on a Virtual Machine, which has the allocation of 4 cores and 4 GB RAM. The monitoring tools Grafana and Prometheus were running as docker containers in the same Virtual Machine.

SBK connecting Pravega for traffic ingesion: *docker run -p 192.168.1.14:9718:9718/tcp kmgowda/sbk:latest -class Pravega -controller tcp://192.168.1.13:9090 -writers 1 -size 1000 -seconds 60*

SBK connecting Pulsar for traffic ingestion: *docker run -p 192.168.1.12:9718:9718/tcp kmgowda/sbk:latest -class Pulsar -broker tcp://192.168.1.13:6650 -partitions 1 -writers 1 -size 1000 -seconds 60*

The SBKs Throughput Mode and End-to-End Latency Modes of Pravega and Pulsar were compared. For each run of both storage systems, the data payload has been changed by ten times and the necessary performance metrics have been obtained. In one set of comparisons, the throughput attained and delay were measured for the 50–90% for single versus ten

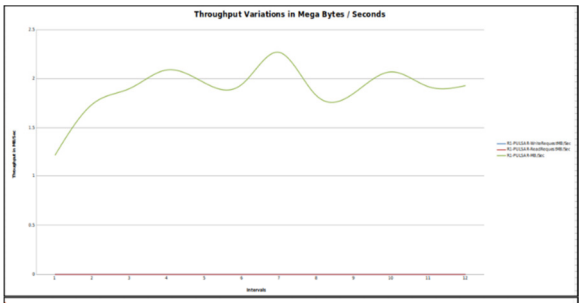
producers. The latency on both storage types was assessed and throughput mode tests were run on 0.5 million records with 10 producers, 10 K bytes of payload, and a consistent throughput of 10 Mbps.

IV. RESULTS AND DISCUSSION

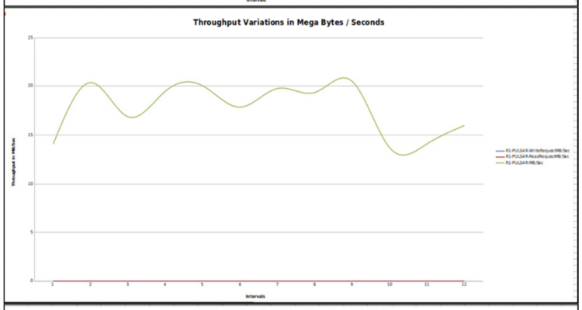
A. *Single Producer: Throughput with Different Data Payload Sizes*

The SBS-chart snippets of the results with Single Producer can be evidenced in Figures 1 and 2.

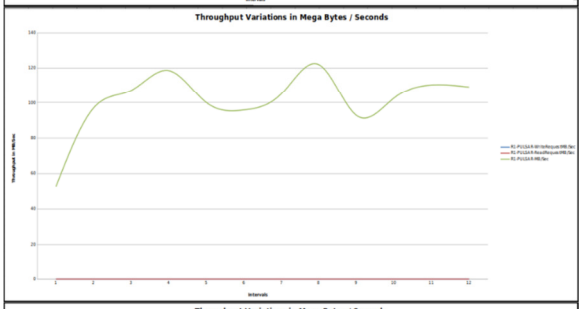
(a)



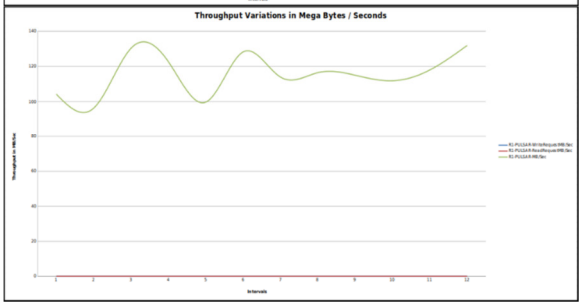
(b)



(c)



(d)



(e)

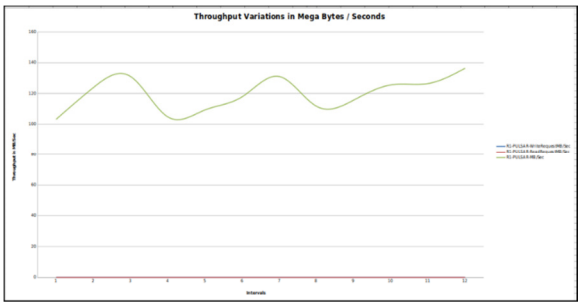
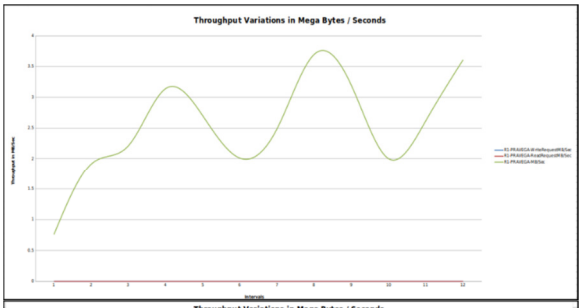
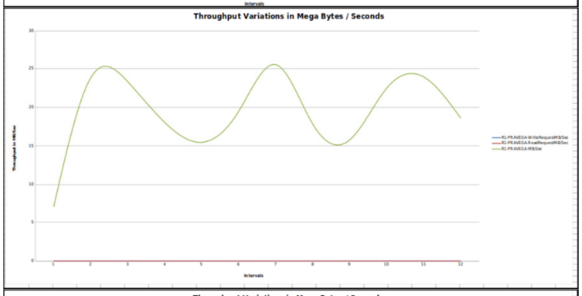


Fig. 2. Throughput of Pulsar single producer data size for (a) 10, (b) 100, (c) 1000, (d) 10000, (e) 100000 bytes.

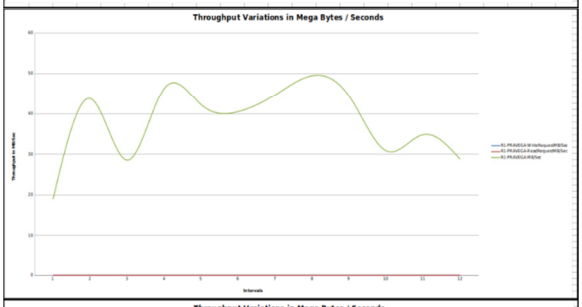
(a)



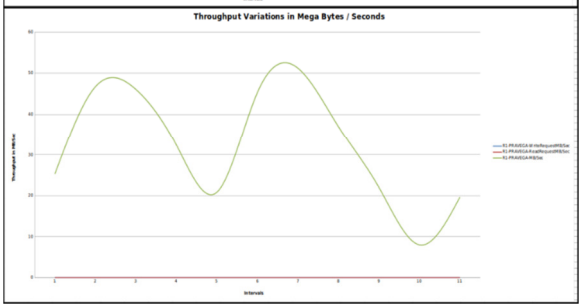
(b)



(c)



(d)



(e)

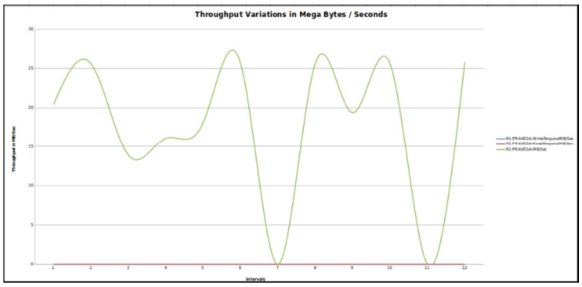


Fig. 3. Throughput of Pravega single producer data size for (a) 10, (b) 100, (c) 1000, (d) 10000, (e) 100000 bytes.

TABLE III. SINGLE PRODUCER THROUGHPUT COMPARISON

Pulsar Throughput (M/s)	Data size	Pravega Throughput (M/s)
2	10	2.67
18.64	100	20
104	1000	42
115	10000	40
122	100000	22

In Figure 4, the y-axis is the throughput in Mbps and the x-axis represents the data payload size in bytes.

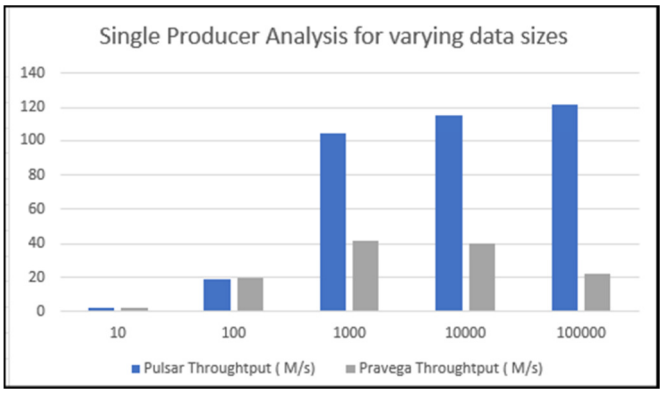


Fig. 4. Single produced analysis.

Figures 2 and 3 indicate that for lower data payload sizes, such as 10 and 100 bytes, the throughput values are almost the same, with Pravega values being slightly higher. When the payload size increases, it can be noticed that the throughput values were being increased more than two to three times. For example, at 100000 payload size, Pulsar gives 122 Mbs compared to the merely around 22 Mbps of Pravega. Clearly the throughput values in the case of Pulsar for higher data payload sizes are better.

B. Single Producer: Latency Comparison with Different Data Payload Sizes

Figures 5 and 6 manifest the latency measurements for 50-90% for different data sizes for Pulsar. Tables IV and V tabulate the results.

(a)

(b)

(c)

(d)

(e)

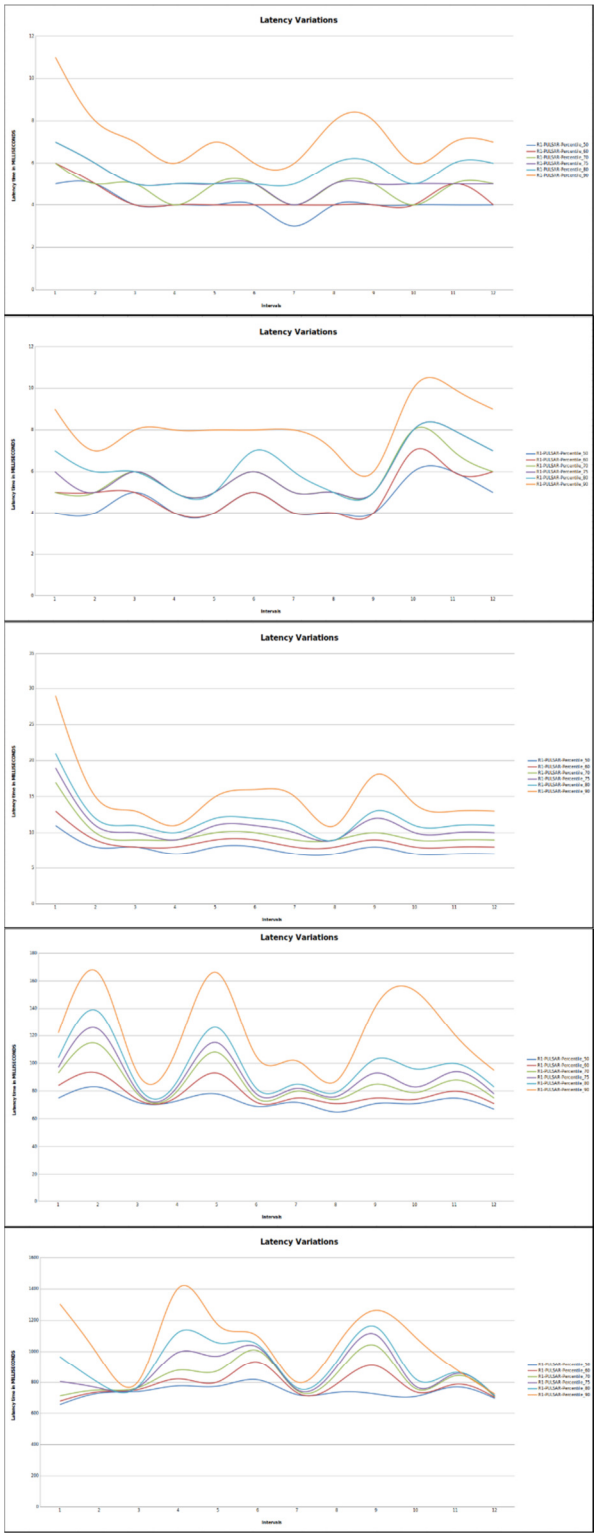


Fig. 5. Latency of Pulsar single producer data size for (a) 10, (b) 100, (c) 1000, (d) 10000, (e) 100000 bytes.

(a)

(b)

(c)

(d)

(e)

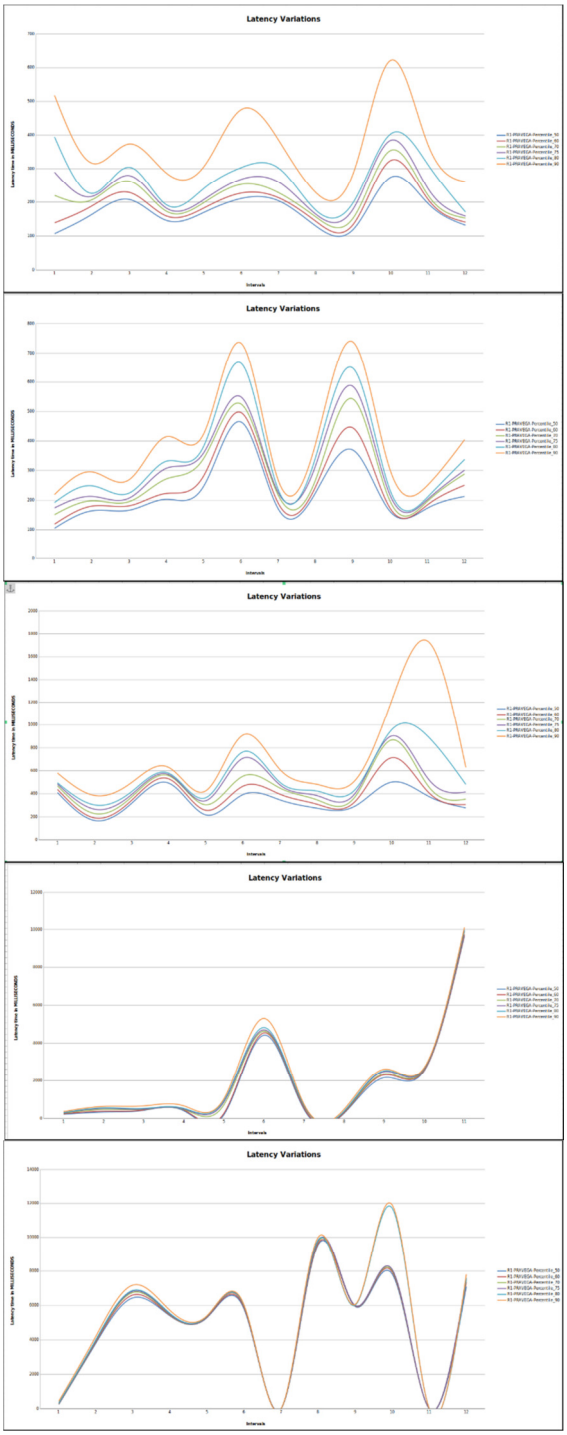


Fig. 6. Latency of Pravega single producer data size for (a) 10, (b) 100, (c) 1000, (d) 10000, (e) 100000 bytes.

TABLE IV. PULSAR SINGLE PRODUCER RESULTS

50%	60%	70%	80%	90%	Data size
4	5	6	8	9	10
4	5	6	8	9	100
8	9	10	12	16	1000
71	74	79	83	153	10000
725	908	1038	1158	1265	100000

TABLE V. PRAVEGA SINGLE PRODUCER RESULTS

50%	60%	70%	80%	90%	Data size
120	135	154	200	288	10
225	245	268	344	390	100
346	396	448	497	608	1000
476	654	683	700	744	10000
5709	6602	6813	6869	7177	100000

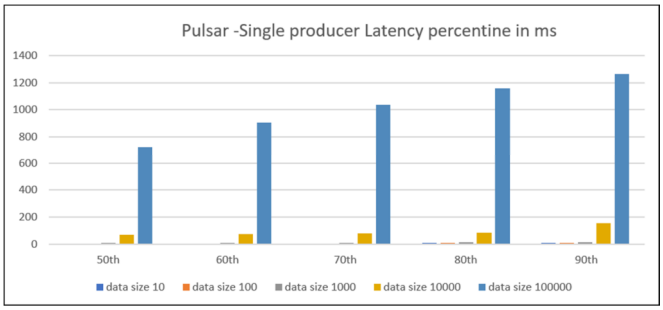


Fig. 7. Single producer latency summary- Pulsar.

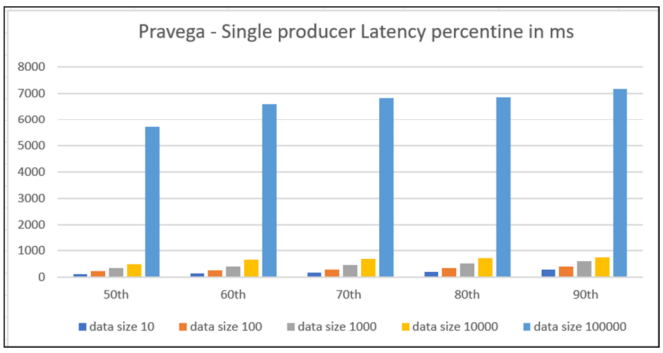


Fig. 8. Single producer latency summary- Pravega.

When assessing the from 50 to 90% levels, the latency values for Pravega are in the range from 120 ms to 600 ms, whereas for Pulsar they are less than 20 ms for smaller data payload sizes, such as 10, 100, and 1000 bytes. The 10000 pulsar latency ranges from 70 to 153 ms, whereas the Pravega latency ranges from 400 to 750 ms. When a payload size of 100,000 is used, the highest latency for Pulsar is 1265 ms, whereas for Pravega it is 7177 ms. One commonality between the two scenarios is that when comparing payload sizes of 10,000 and 100,000, there is a significant increase in both cases. It is evident that the latency numbers of Pulsar are significantly better for all data payload sizes.

C. Ten Producers: Throughput with Different Data Payload Sizes

Tests were conducted with different data payload sizes with 10 producer configurations on Pravega and Pulsar storage systems. Figures 9 and 10 and Table VI display the results. In Figure 11, the y-axis represents the throughput in Mbps and the x-axis denotes the data payload size in bytes.

As it would be expected, Figures 9-10 reveal that the throughput rose for numerous producers. This anticipated behavior is seen in the case of Pulsar, however when Pravega is taken into account, the numbers drop when compared to single producer throughput figures. Another finding is that throughput

is lower for payload sizes of 100,000 than it is for payload sizes of 10,000 in the Pulsar instance. A minor rise appears to have occured in Pravega's speed ranging from 28 Mbps to 36 Mbps. It can be stated that Pravega performs marginally better in multi-producer cases, but only for very large payload sizes.

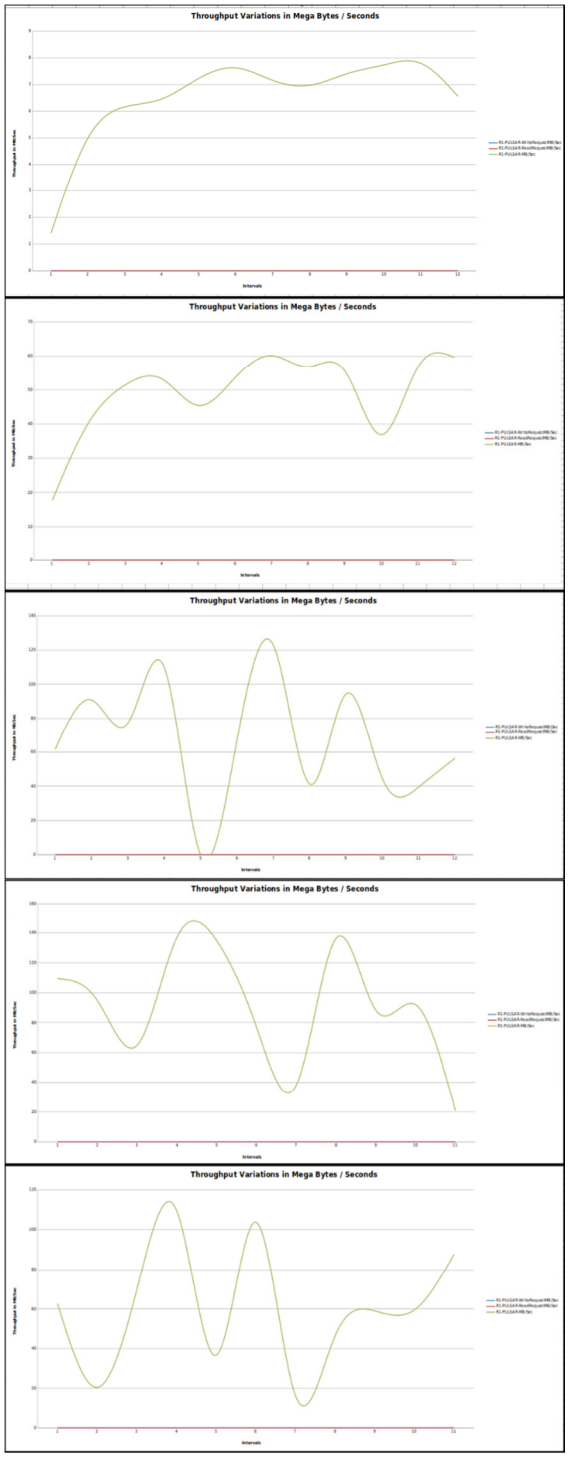


Fig. 9. Pulsar 10 producers throughput data size for (a) 10, (b) 100, (c) 1000, (d) 10000, (e) 100000 bytes.

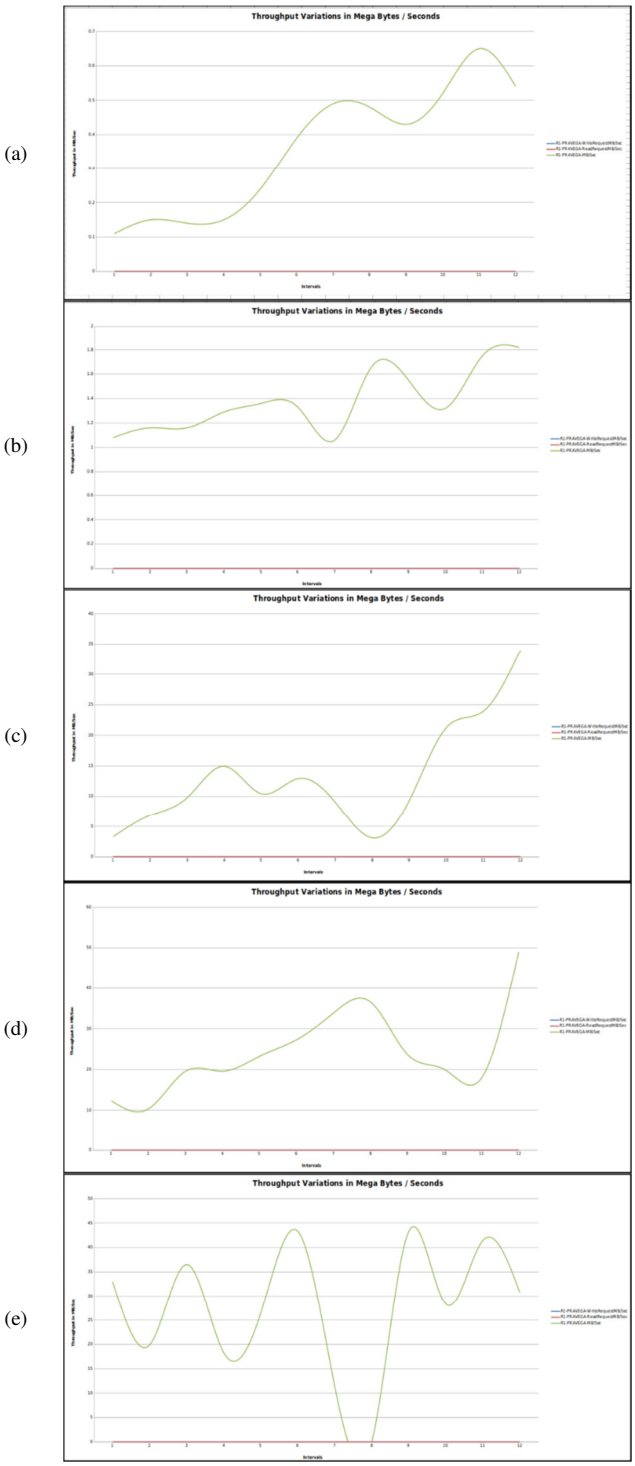


Fig. 10. Pravega 10 producers throughput data size for (a) 10, (b) 100, (c) 1000, (d) 10000, (e) 100000 bytes.

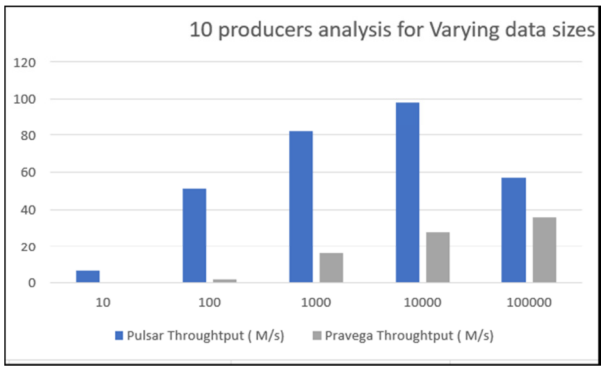


Fig. 11. Ten producers throughput summary.

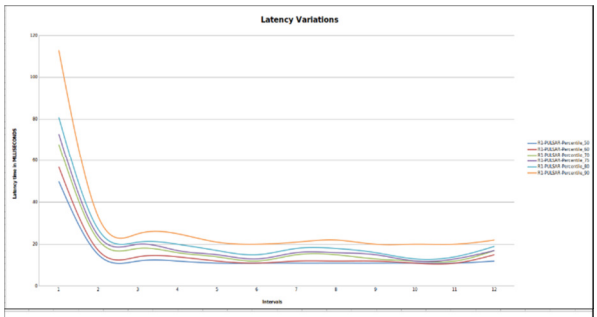
TABLE VI. TEN PRODUCER THROUGHPUT COMPARISON

Pulsar Throughput (M/s)	Data size	Pravega Throughput (M/s)
6.67	10	0.45
51	100	1.6
82	1000	16
98	10000	28
57	100000	36

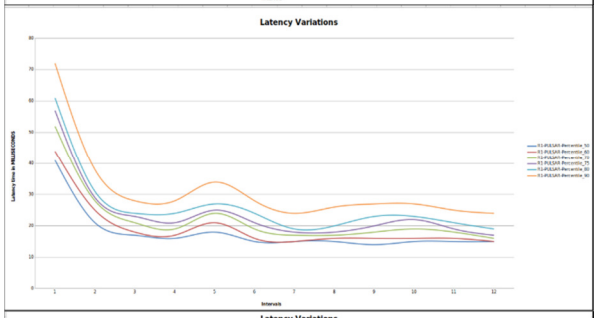
D. Ten Producers: Latency with Different Data Payload Sizes

The results can be evidenced in Figures 12-15 and Tables VII-VIII.

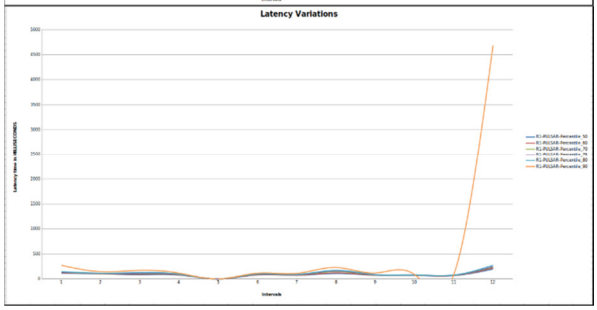
(a)



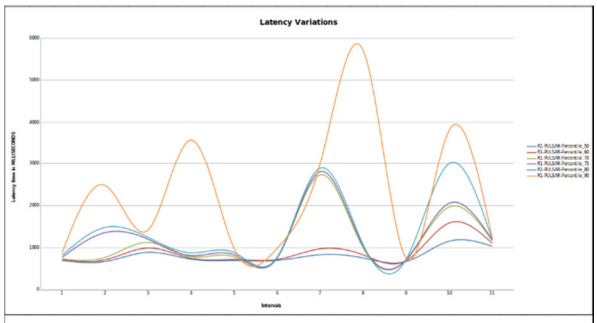
(b)



(c)



(d)



(e)

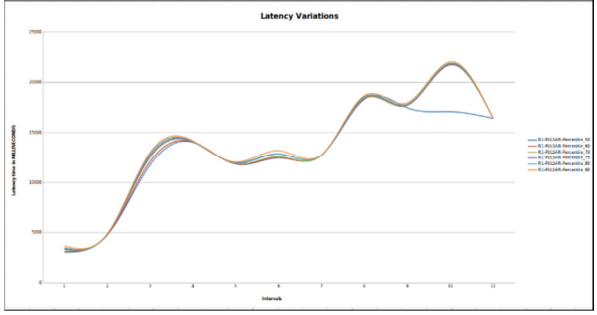
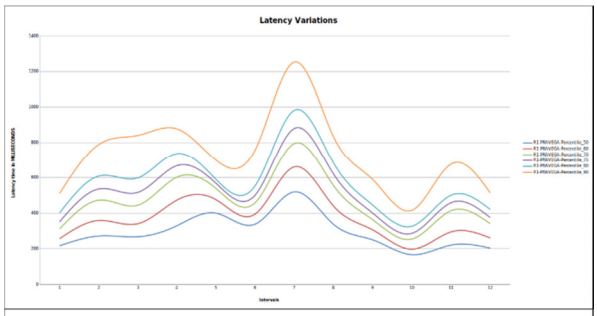
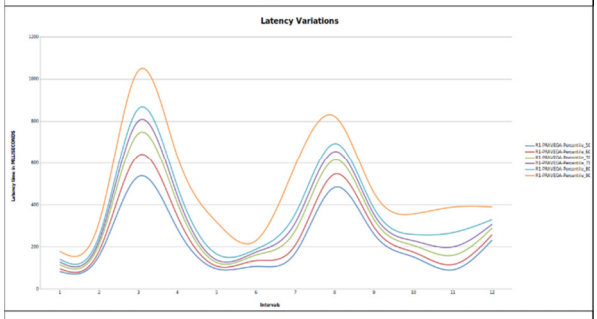


Fig. 12. Pulsar 10 producers latency. Data size: (a) 10, (b) 100, (c) 1000, (d) 10000, (e) 100000 bytes.

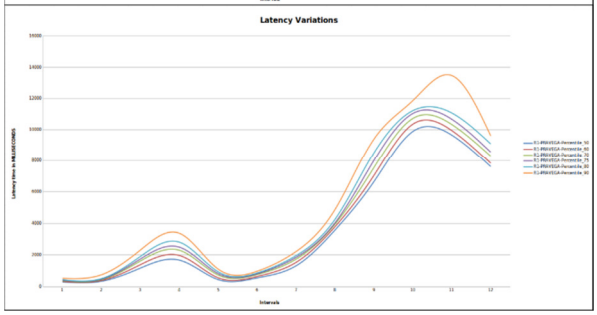
(a)



(b)



(c)



(d)

(e)

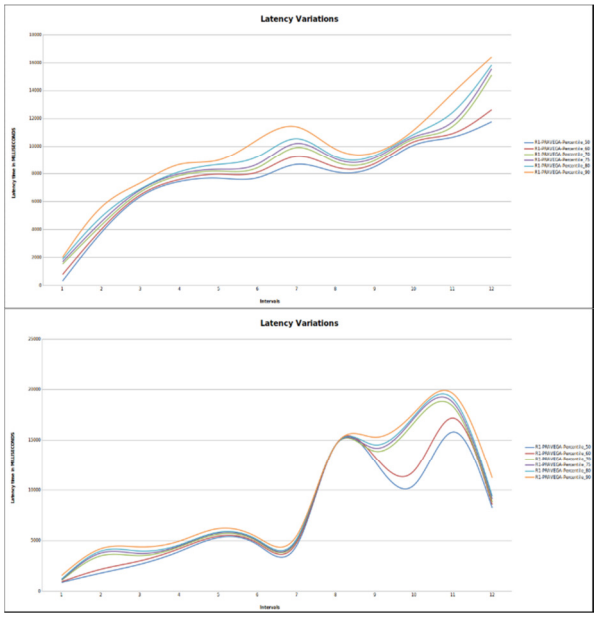


Fig. 13. Pravega 10 producers latency. Data size: (a) 10, (b) 100, (c) 1000, (d) 10000, (e) 100000 bytes.

TABLE VII. PULSAR 10 PRODUCER LATENCY

50%	60%	70%	80%	90%	Data size
11	12	15	18	22	10
14	16	19	23	27	100
108	121	146	158	171	1000
721	780	829	884	1245	10000
12600	12870	12901	13207	14186	100000

TABLE VIII. PRAVEGA 10 PRODUCER LATENCY

50%	60%	70%	80%	90%	Data size
11	12	15	18	22	10
14	16	19	23	27	100
108	121	146	158	171	1000
721	780	829	884	1245	10000
12600	12870	12901	13207	14186	100000

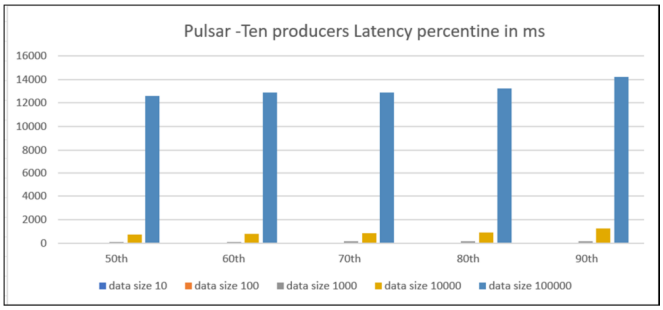


Fig. 14. Pulsar 10 producers latency summary.

For smaller data payload sizes (10, 100, and 1000 bytes), the latency values of Pulsar are between 11 and 170 ms, whereas those of Pravega are between 3300 and 4800 ms. According to Figures 14 and 15, pulsar latency for 10,000 data size is between 700 and 1200 ms, whereas for Pravega, it is from 8000 to 11000 ms. When a payload size of 100,000 is used, the maximum latency for Pulsar is 14,000 ms, whereas for Pravega it is 17,000 ms. Clearly, Pulsar's latency statistics are significantly superior than those of Pravega for all data payload sizes.

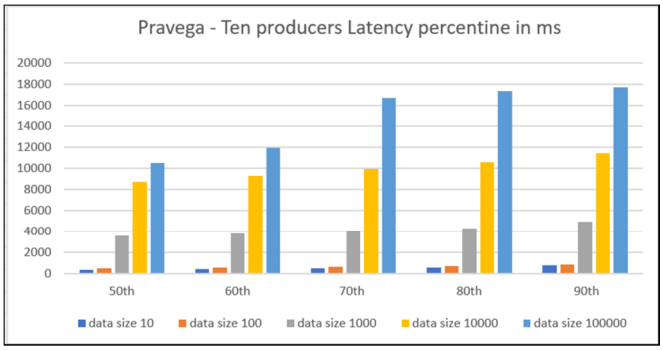


Fig. 15. Pravega 10 producers latency summary.

E. Latency for 0.5 Million Record Writes

Tests were carried out writing 0.5 million records at a rate of 10 Mbps while the data payload size was being varied. Charts and documented percentile values for Pulsar and Pravega with 10 producers and connection pooling enabled can be observed in Figures 16-24.

Figure 16 shows that the latency levels of Pulsar are very low for the 5- 50 percentile levels (around 5 ms). Figure 17 discloses that the latency slightly increases to 20 ms until the 95 percentile and reaches 160 ms for the 99.99 percentile levels. The same can also be pinpointed in Figures 18 and 19 where 5 s intervals were captured for all the percentile levels in the Pulsar case.

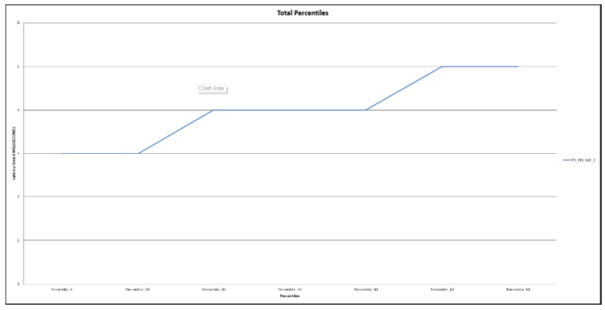


Fig. 16. Pulsar latency 5 to 50 percentile.

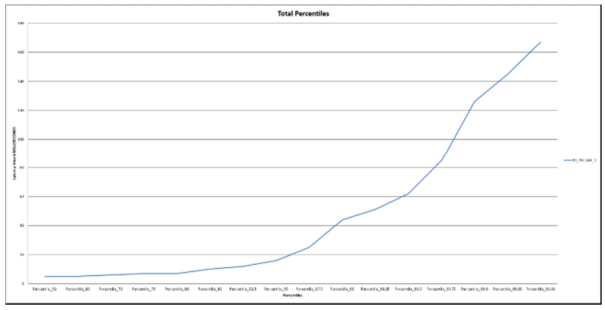


Fig. 17. Pulsar latency 50 to 99.99 percentile.

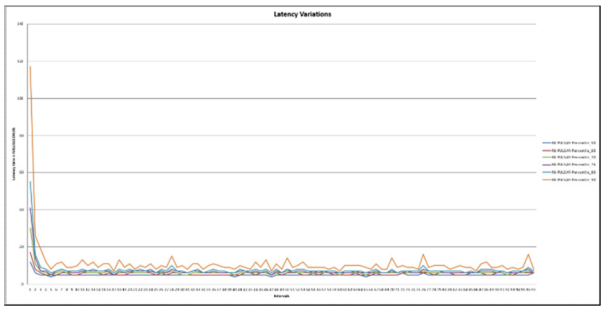


Fig. 18. Pulsar latency 5 to 50 percentile per 5 s intervals.

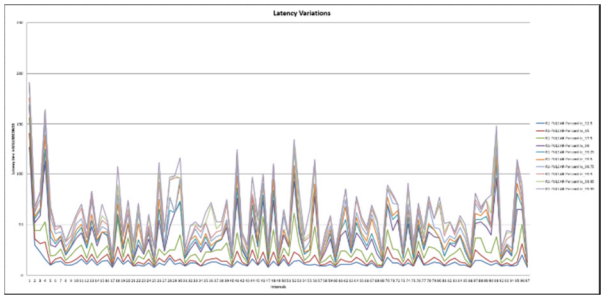


Fig. 19. Pulsar latency 90 to 99.99 percentile for 5 s intervals.

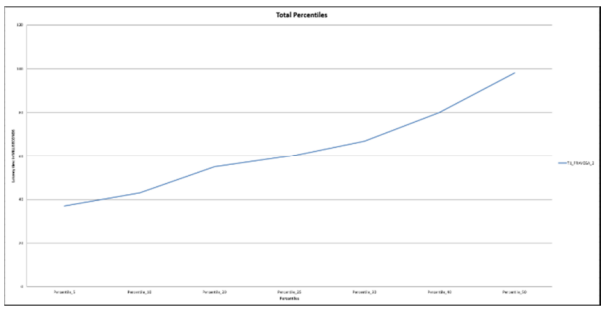


Fig. 20. Privega latency 5 to 50 percentile.

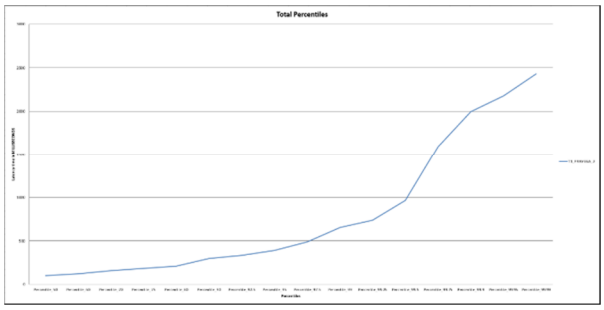


Fig. 21. Privega latency 50 to 99.99 percentile.

Figure 20 demonstrates that the 5-50 percentile latency levels of Pravega are extremely low, ranging from 40 to 100 ms. As can be seen in Figure 21, the latency increases marginally to 400 ms at the 95 percentile and exceeds 2000 ms at the 99.99 percentile. The same is noticed in Figures 22 and 23, where 5 s intervals were recorded for each percentile level in the Pravega scenario. It is evident that Pravega has significantly greater latency levels than Pulsar when writing 0.5 million records at a 10 Mbps writing speed. Connection pooling was enabled in each of the aforementioned tests.

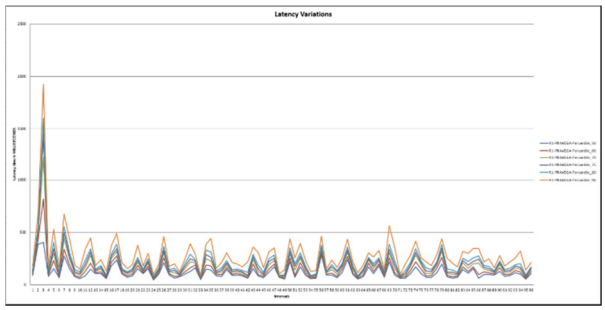


Fig. 22. Pravega latency 50 to 90 percentile for 5 s intervals.

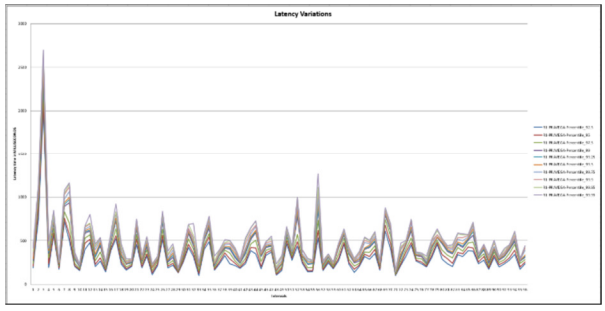


Fig. 23. Pravega latency 90 to 99.99 percentile for 5 s intervals.

V. CONCLUSION

The current study presents a comparison of two open source distributed streaming storage systems: Pravega and Pulsar. The Common Storage Benchmarking Kit allowed us to run a variety of test categories, including latency measurements for writing 0.5 million records at a 10 Mbps injection rate, both for single and multiple producer cases, and for different data payload sizes. Regarding the resulting data metrics plotted onto charts, suitable conclusions can be drawn from each case once the results are analyzed. While connection pooling improves system resilience, it also somehow lowers system performance. Authors in [15] examined how connection pooling affects Pravega storage. For the purpose of this study's comparative analysis of the two storage systems, connection pooling was enabled. The system load in which each distributed storage system performs better is the focus of this study. To mitigate the performance decrease that occurs when connection pooling is enabled, more research must be considered as a part of future research.

REFERENCES

[1] "Pravega – A Reliable Stream Storage System." https://cncf.pravega.io/.

[2] "pravega/pravega," https://github.com/pravega/pravega.

[3] "Apache Pulsar." https://pulsar.apache.org/.

[4] N. V. Sanjay Kumar and K. Munegowda, "Distributed Streaming Storage Performance Benchmarking: Kafka and Pravega," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 2S, pp. 1–8, Dec. 2019, https://doi.org/10.35940/ijitee.B1001.1292S19.

[5] "Release Storage Benchmark Kit Version 5.0 · kmgowda/SBK," *GitHub*. https://github.com/kmgowda/SBK/releases/tag/5.0.

[6] "Dashboards | Grafana documentation," *Grafana Labs*. https://grafana.com/docs/grafana/latest/dashboards/.

[7] "Apache BookKeeper." https://bookkeeper.apache.org/.

[8] "Apache ZooKeeper." https://zookeeper.apache.org/.

[9] F. Junqueira and B. Reed, *ZooKeeper: Distributed Process Coordination*. Sebastopol, CA, USA: O'Reilly, 2013.

[10] "apache/flink," https://github.com/apache/flink.

[11] "apache/samza." https://github.com/apache/samza.

[12] N. Sajitha and S. P. Priya, "Optimal Artificial Neural Network-based Fabric Defect Detection and Classification," *Engineering, Technology & Applied Science Research*, vol. 14, no. 2, pp. 13148–13152, Apr. 2024, https://doi.org/10.48084/etasr.6773.

[13] T. Alshammari, "Using Artificial Neural Networks with GridSearchCV for Predicting Indoor Temperature in a Smart Home," *Engineering, Technology & Applied Science Research*, vol. 14, no. 2, pp. 13437–13443, Apr. 2024, https://doi.org/10.48084/etasr.7008.

[14] H. T. S. Alrikabi, I. A. Aljazaery, and A. H. M. Alaidi, "Using a Chaotic Digital System to Generate Random Numbers for Secure Communication on 5G Networks," *Engineering, Technology & Applied Science Research*, vol. 14, no. 2, pp. 13598–13603, Apr. 2024, https://doi.org/10.48084/etasr.6938.

[15] K. V. Ramesh and G. T. Raju, "Pravega: Performance impact analysis with Connection Pooling'," in *2nd IEEE International Conference on Knowledge Engineering and Communication Systems (ICKECS 2024)*, Karnataka, India, Apr. 2024.