Distributed Streaming Storage Performance Benchmarking: Kafka and Pravega

Sanjay Kumar N V, Keshava Munegowda

Abstract: ***The performance benchmarking tool for a distributed streaming storage system should be targeted to achieve maximum possible throughput from the streaming storage system by thrusting data massively. This paper details the design and implementation of high-performance benchmark tool for Kafka and Pravega streaming storage systems. The benchmark tool presented in this paper supports multiple writers and readers. The Pravega streaming storage is evaluated against Kafka with respect to performance.***

Keywords: ***Benchmarking, Big Data, Concurrency, Distributed Systems, Events, Kafka, Latency, Open Messaging, Performance, Pravega, Streams, Storage, Throughput.***

I. INTRODUCTION

Apache Kafka [1], [2] is one of the widely used distributed streaming [3] storage systems. An existing Kafka (version 2.3.0) producer benchmark tool [4], [5] for write performance benchmarking and consumer benchmark tool [4], [5] for read performance benchmarking, supports only one producer/writer and consumer/reader respectively. But, in a real-time scenario/use case, there will be multiple applications flushing or reading data to/from a single Kafka client. So, it's always better to exercise the Kafka client or any streaming system with multiple producers and consumers. If the single instance of the benchmark tool allows the multiple producers/writers, then these multiple threads need to be synchronized while writing data to Kafka client and also aggregating the data write responses from multiple threads. However, the synchronization of multiple threads reduces the strength of the benchmark tool and it won't be able to flush more data to Kafka client.

In this paper, we present the design and implementation details of Pravega [6], [7] performance benchmark tool [8]. The Pravega benchmark tool was initially developed for the performance benchmarking of Pravega streaming storage and later got extended for Kafka performance benchmarking too. The key differentiation of Pravega benchmark tool compares to Kafka producer and consumer benchmarking tools are

- Supports multiple producers and consumers.
- A common tool for both writer benchmarking and reader benchmarking.
- Supports End to End latency.
- Latency percentile calculations are performed for all the events/messages without any sampling

Even though, the Pravega benchmarking tool supports multiple writers/readers, it does not compromise on the speed at which data is flushed to Kafka client/Pravega client. The Open Messaging benchmark [9] tool also supports Kafka's performance benchmarking with multiple producers and consumers. In this paper, we evaluate the Pravega benchmark tool against the Kafka producer benchmark tool for single producer benchmarking, Kafka consumer benchmark tool for single consumer and Open messaging benchmark tool for performance benchmarking of multiple producers. The proposed design and implementation of benchmarking tool of this paper is used for comparison of Kafka and Pravega for single producer/consumer and multiple producers/consumers performance benchmarking. The same tool is used for End to End Latency benchmarking of both Kafka and Pravega.

II. DESIGN OF KAFKA AND PRAVEGA PERFORMANCE BENCHMARK TOOL

As part of performance benchmarking, before sending an event/message, the writer/producer records the start time by using API System.currentTimeMillis() [10] and once the write is acknowledged then the response thread records the end time by using the same API (Application Programming Interface). We have experimented using the other API System.NanoTime() [10] and class Instant APIs [10], but the System.currentTimeMillis() API has proven very fast and thread safe too. The time precision of System.currentTimeMillis() is in milliseconds is enough for stream storage benchmarking.

Both Kafka and Pravega provide the client APIs for streaming data into their clusters and reading the same. If the benchmark tool supports multiple producers and/or consumers, then multi-threads synchronization is required while accessing the client APIs and consolidating the read/writer responses from Kafka/Pravega cluster. The first case is inevitable, and it should be taken care of by Kafka client or Pravega client. But, the second case is the benchmark tool's responsibility. In our experiments, it was evident that the usage of synchronized [10] method for multiple thread synchronization reduces the strength of the benchmark tool and thus not be able to flush/read more events to the Kafka/Pravega client. Hence, we designed the architecture of the benchmark tool with a shared message queue as shown in Fig. 1.

Revised Manuscript Received on December 12, 2019.

Mr. Sanjay Kumar N V, Associate Professor, Department of CSE, Kalpataru Institute of Technology, Tiptur, INDIA, sanjaynv@gmail.com

Dr. Keshava Munegowda, Consultant, Pravega Bangalore, INDIA, keshava.gowda@gmail.com

Upon receiving the response to write/read, each of the response threads enqueue the start time and end time of read/write operation; on the other end, a dedicated benchmark processing thread dequeue this information for further throughput and latency calculations. We tried Java 8's Array Blocking Queue [11], [12], Linked Blocking Queue [11], [13] and Concurrent Linked Queue [11], [14], but the Concurrent Linked Queue does very well with respect to performance and its implementation is based on wait-free algorithm described by Maged M. Michael and Michael L. Scott [15]. But the catch here it is, the poll [14] method of this concurrent Linked queue is non-blocking. This means, if the benchmark processing thread keeps invoking poll method in a tight loop causes the synchronization with add [14] method and it again reduces the strength of the benchmark tool; So, to mitigate this problem, if there are no items in the concurrent queue then the benchmark processing thread does busy waiting for a constant time (1 millisecond, in our test setup). Another observation is, this benchmark processing thread should not sleep, because waking up and scheduling a thread are bit time consuming operations.

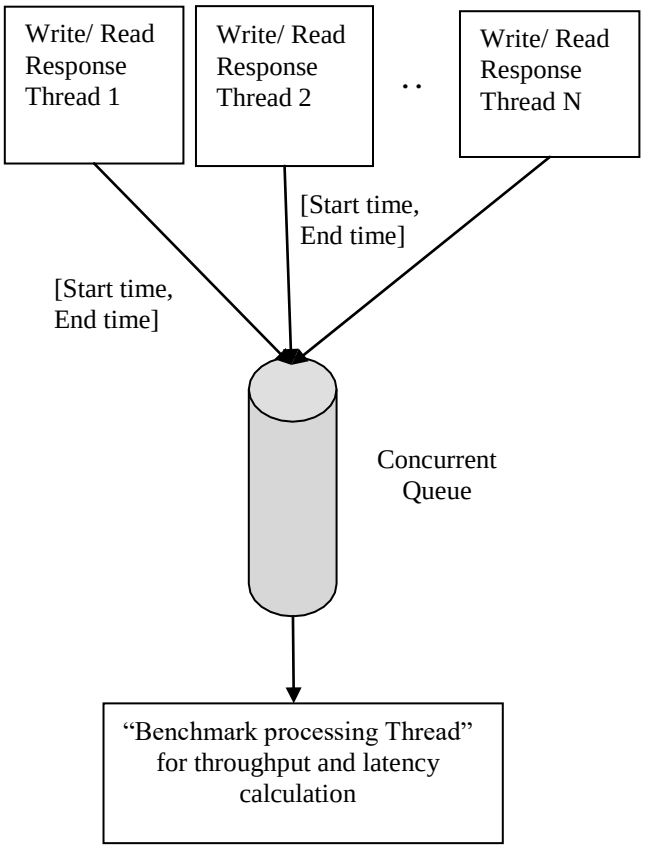

Fig. 1: Design of Pravega benchmark tool

For throughput calculations, the benchmark processing thread counts each of the events received from the concurrent queue; the throughput is calculated and published for configured for specific window intervals. To calculate the time window intervals, the benchmark processing thread uses time instant [10] API to avoid synchronizations with System.currentTimeMillis() API which is used by write/read response threads.

For latencycalculations, the time duration between start time and end time for each of the event is termed as latency; Note that, end time gets recorded once the response thread is scheduled, so scheduling time of response time is inclusive of latency measurement. If the system, in which the benchmark tool is running, is heavily loaded then the latency value may go higher because the scheduling time of response thread was more. A dedicated integer array is used to keep track of latency values. The latency value is used as an index of an array and the same index value is incremented to track the latencies received. This approach is inspired by the counting sort algorithm and the latency array is always sorted even while recording the latencies. The proportional index values are picked for the latency percentile calculations.

Another important observation is the byte array serialization and deserialization are faster than compare to string serialization and deserialization; so, the producers/consumers send/read the data in the form of array of bytes only.

III. IMPLEMENTATION DETAILS

The described high-performance design of the benchmark tool is implemented in Java 8 and is available at this Git Hub [8]:
https://github.com/kmgowda/pravega-benchmark/releases/tag/v1.0 .

Since this code is open source and free; so, interested developers can also review the same, raise issues and contribute code for further improvements.

The Pravega benchmark tool provides the following modes of the Benchmarking.

i) Burst Mode
ii) Throughput Mode
iii) OPS Mode or Events Rate/Rate limiter Mode
iv) End to End Latency Mode

i) Burst Mode

In this mode, the Pravega benchmark tool puts the heavy load on the system by pushing/pulling the messages to/from the Pravega/Kafka client as much as possible. This mode is used to find the maximum throughput that can be obtained from the Pravega/Kafka cluster. This mode can be used for both producers and consumers. This mode is used for all the throughput comparisons presented in this paper.

ii) Throughput Mode

In this mode, the Pravega benchmark tool pushes the messages to the Pravega/Kafka client with specified approximate maximum throughput in terms of Mega Bytes/second (MB/s). This mode is used to find the least latency that can be obtained from the Pravega/Kafka cluster for a given throughput. This mode is used only for write operation and useful for analyzing throughput vs latency.

iii) OPS Mode or Events Rate/ Rate limiter Mode

This mode is another form of controlling writer's throughput by limiting the number of events per second. In this mode, the Pravega benchmark tool pushes the messages to the Pravega/Kafka client with specified approximate maximum events per sec. This mode is used to find the least latency that can be obtained from the Pravega/Kafka cluster for events rate. This mode is used only for write operation.

iv) End to End Latency Mode

In this mode, the Pravega benchmark tool writes and read the messages to the Pravega/Kafka cluster 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 producers and consumers.

IV. RESULTS AND DISCUSSION

A. Pravega benchmark tool vs Kafka producer performance tool vs Open messaging Benchmark

The Pravega benchmark tool presented in this paper is expected to yield the same performance as the Kafka producer benchmark tool for a single writer. The hardware and software configurations of these experiments are described in Table I. The Kafka server configuration and topic configuration values used are listed in Tables II and III respectively.

Table I: Hardware and software configuration of Test setup

Components	Remarks
Number of nodes	4 Nodes 1 for Kafka client /Pravega client 3 for Kafka brokers/Pravega segment servers. Each node is installed with RHEL 7.4 release version
CPU	48 CPUs Each of 64 Bit, 2.3 GHz
RAM	186 GB (Giga Bytes)
Hard Disk	SSD (Solid State Drive) 300 MB/s (Mega Bytes/Second) of XFS File system throughput.
Ethernet	10Gbps (Giga Bits/second) Network

Table II: Kafka Configurations

Kafka Components/parameters	Remarks
Version	2.3.0
Number of brokers	3
Number of clients	1
File System	XFS on SSDs
Zookeeper [16] [17]	Version 3.5.4 beta

Table III: Kafka Topic Configurations

Kafka Topic parameters	Remarks
Partitions	15
Replication factor	3
Minimum in sync replicas	2
Idempotence producers	Enabled
Acks	All (-1)
Compression	None
Batch size	16K
Buffer memory	32MB
Serialization	Byte Array
Deserialization	Byte Array
Maximum poll records for consumers (max.poll)	1

The throughput differences for data sizes of 10, 100, 1000, 10000 and 100000 bytes for a single producer is shown in Fig. 2. The idempotent producers are used for write benchmarking. Note that, the difference between maximum throughput values in terms of MB/s (Mega Bytes per Second) achieved by Pravega benchmark tool and Kafka performance benchmark tool is negligible with the added advantage of multi producers support in the pravega benchmark tool. The open messaging benchmark tool records low throughput for data size 10 and 100 bytes; for higher data sizes, the open messaging benchmark tool produces better throughput and these throughput values are comparable with Kafka producer benchmark tool and Pravega benchmark tool. Fig. 3. Shows the same throughput comparison in terms of events/second or ops (operations) /second.

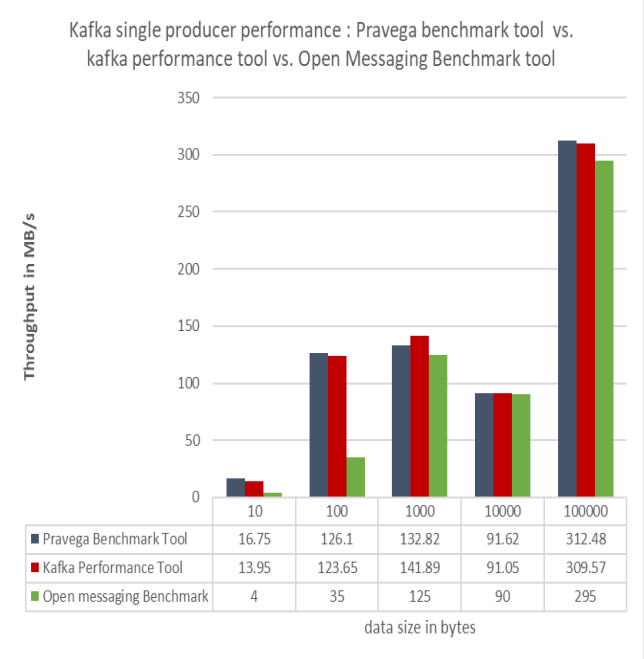

Fig. 2: Pravega benchmark tool vs Kafka benchmark tool vs Open Messaging Benchmark; Single producer performance in terms of MB/s

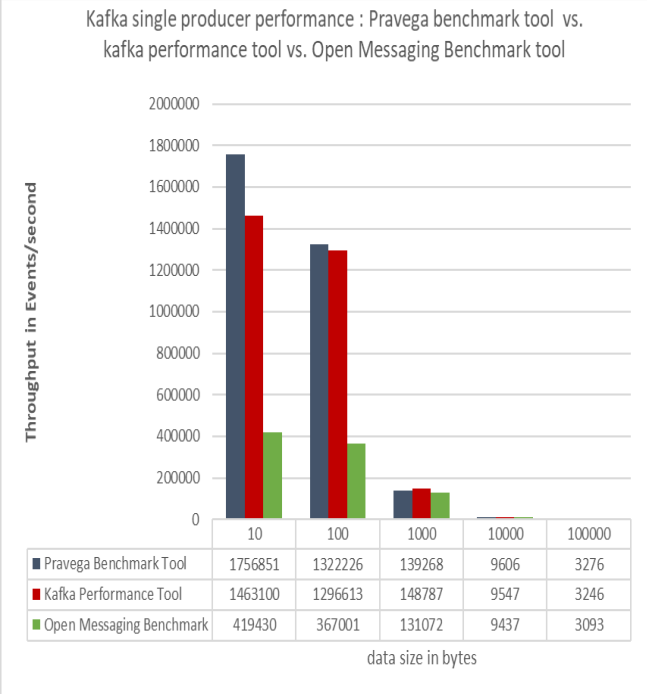

Fig. 3: Pravega benchmark tool vs Kafka benchmark tool vs Open Messaging Benchmark; Single producer performance in terms of

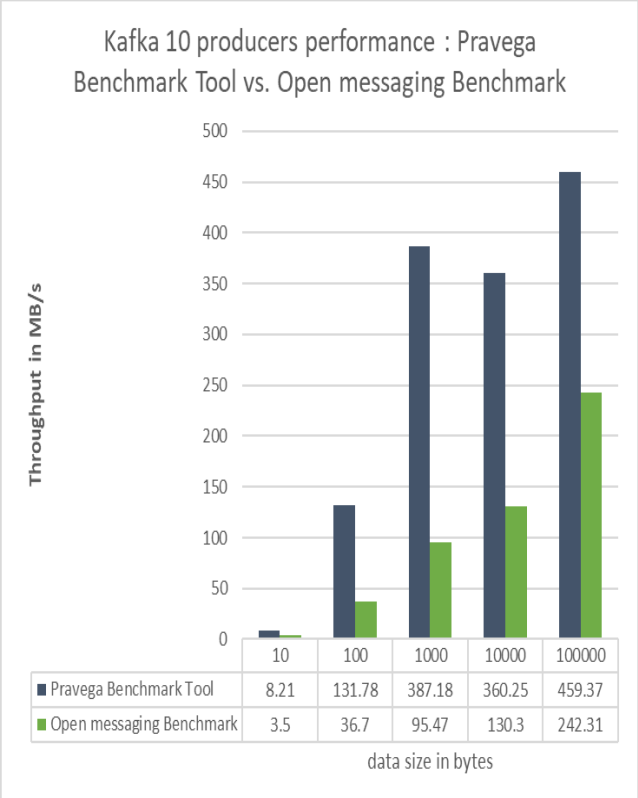

Fig. 4: Pravega benchmark tool vs. Open Messaging Benchmark; 10 Kafka producers performance comparison in terms of MB/s

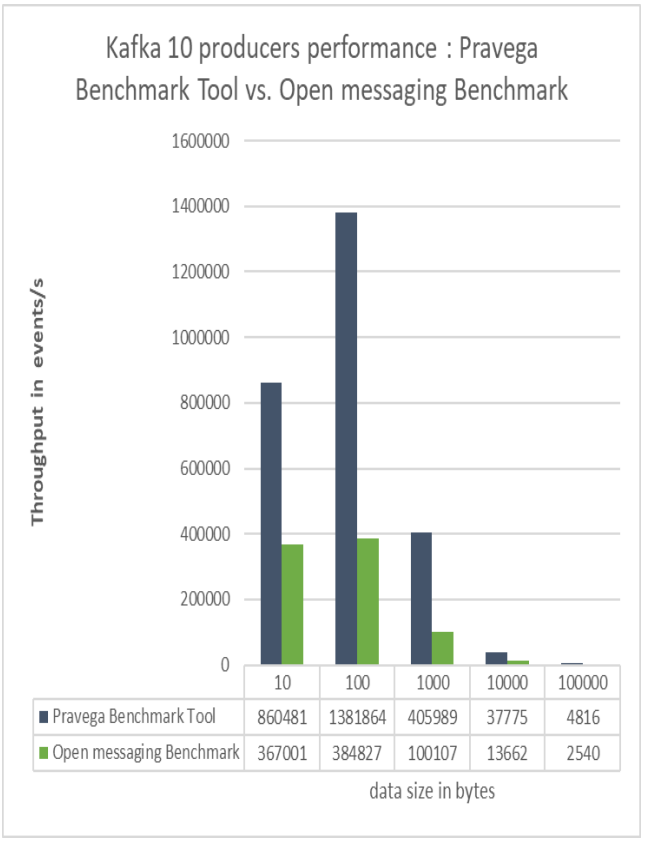

Fig. 5: Pravega benchmark tool vs. Open Messaging benchmark; 10 Kafka producers performance comparison in terms of events/second

Fig. 4. Shows the performance of the 10 Kafka producers using the Pravega benchmark tool and Open messaging benchmark tool. Note that, the throughput achieved by the Open messaging benchmark is very low compared to the Pravega benchmark tool. The open messaging benchmark tool does not pump/flush data to Kafka client at the maximum possible rate. Fig. 5. Shows the same throughput comparison between Pravega benchmark tool and Open messaging benchmark tool but in terms of events/second.

B. Pravega benchmark tool vs Kafka Consumer performance tool

Both the Pravega benchmark tool and Kafka consumer performance tools are producing identical results; The results are shown in Fig. 6. Note that, the compare to write, the read performance is recorded low; this is because the configuration value for maximum number of events to poll (max.poll) is set to 1; the consumer/reader throughput improves if this value is set to high.

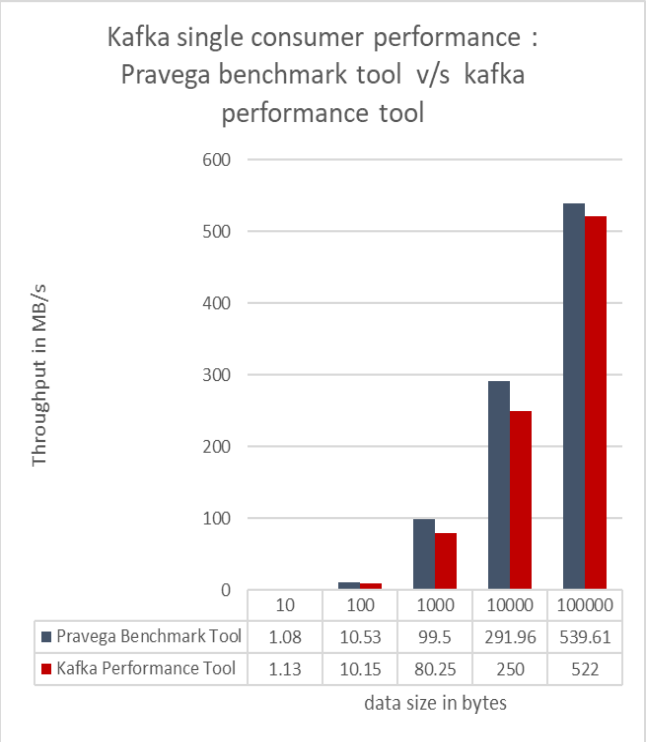

Fig. 6: Pravega benchmark tool vs Kafka benchmark tool; Single Kafka consumer throughput performance in terms of MB/s

C. Kafka vs Pravega performance Benchmarking

It is demonstrated that the Pravega benchmark tool produces the same performance as Kafka benchmarking tool; Now, we can use the Pravega benchmark tool to compare the performance between Kafka and Pravega with single producer and with multiple producers. The Pravega configuration values are listed in Table IV. The pravega segment store/server configurations values are listed in Table V. The Pravega adheres to the Lambda architecture [18] hence for the data durability its first written to Tier 1 storage (typically bookkeeper) and asynchronously shifted to Tier 2 storage.

Table IV: Pravega Configurations

Pravega Components/parameters	Remarks
Version	0.5
Number of segment stores	3
Number of controllers	1
Number of clients	1
File System	XFS on SSDs
Zookeeper	Version 3.5.4 beta
Tier 1 Storage	Bookkeeper [19] [20], Version 4.7.3 3 bookies are used. One bookie per segment store
Tier 2 Storage	Hadoop [21] [22] version 2.7.3. 3 nodes cluster.

Table V: Pravega segment store configurations

Pravega Segment store configuration parameters	Remarks
Segment containers	32
Cache Size	64 GB (GigaBytes)
Rocks DB [18] [23] write cache size	16 GB
Rocks DB read cache size	512 MB (MegaBytes)
Rocks DB cache block size	32 KB (KiloBytes)
Tier 2 Writer flush bytes	64 MB

1) Single producer Benchmarking

Fig. 7. Shows the single producer/writer performance difference between Kafka and Pravega. Note that the Kafka writes are very fast compare to Pravega; but, if the log flush parameter (log.flush.interval.messages set to 1) set to flush every event to disk to improve the durability, then performance degrades as shown in same Fig. 7. The Pravega performance depends on the Tier1 (bookkeeper) performance, rocks dB performance and the cache settings of the segment stores.

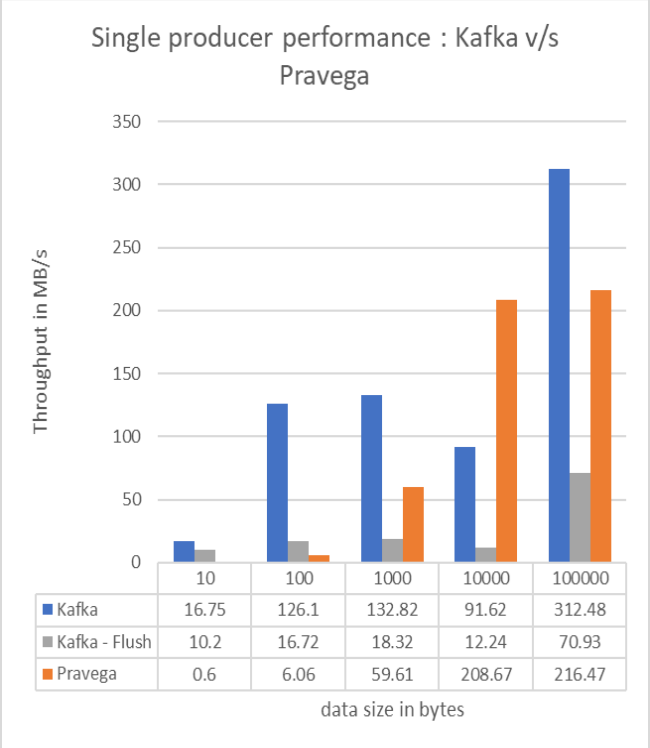

Fig. 7: Single producer performance benchmarking: Kafka vs. Pravega

2) Single Consumer Benchmarking

Fig. 8. Shows the performance difference between a single consumer of Kafka and Pravega. Whenever the Pravega read involves Tier 2 storage read, the performance degrades.

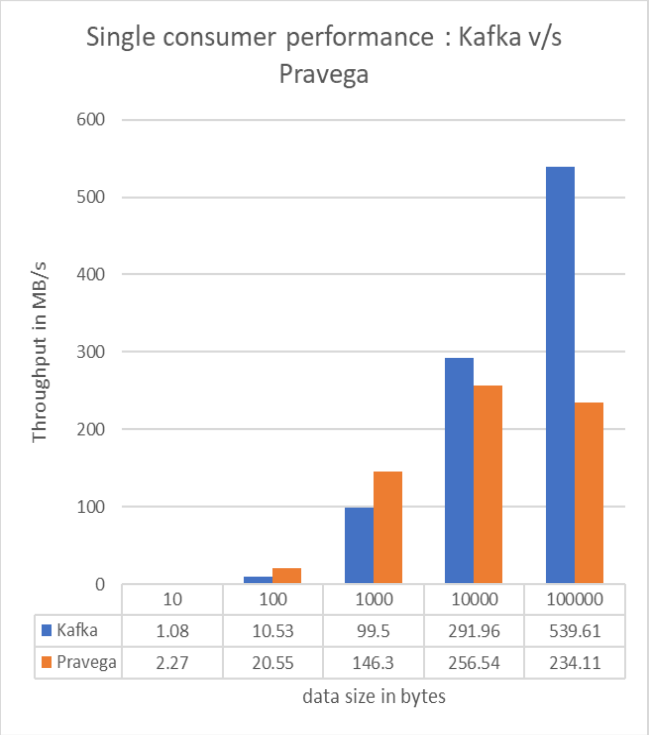

Fig. 8: Single consumer performance benchmarking: Kafka vs. Pravega

3) Multiple producers Benchmarking

Fig. 9. shows the 10 producers/writers performance difference between Kafka and Pravega. Note that the Kafka writes are very fast compare to Pravega; but, if the log flush parameter (log.flush.interval.messages set to 1) set to flush every event to disk to improve the durability, then performance degrades.

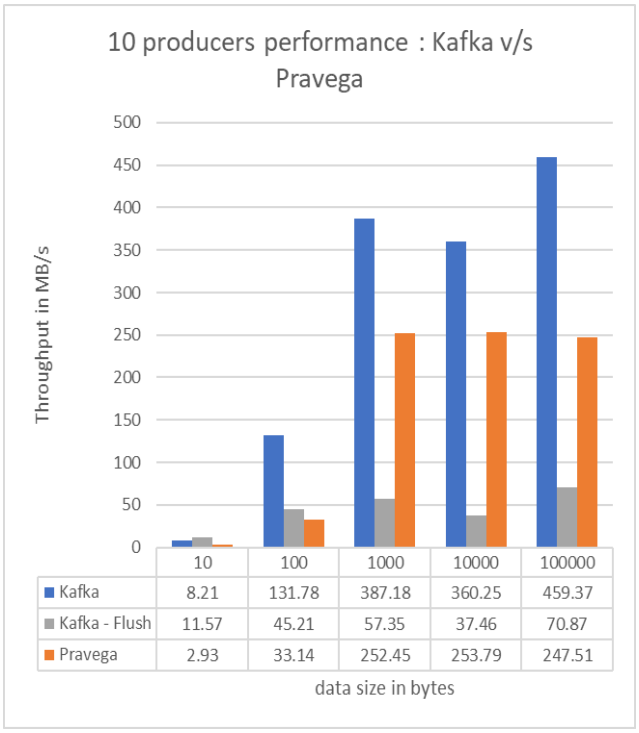

Fig. 9: 10 producers performance benchmarking: Kafka vs. Pravega

4) Throughput vs Latency

Fig. 10, 11 and 12 shows the 50th (median), 75th and 99th percentile of latencies for throughput range 10, 20, 30, 40 and 50 MB/S with 10 producers/writers. If the flushing enabled for Kafka for every single event, the latency of Kafka goes higher than 100 MS(milliseconds); so, in below figures, the flush is set to default value (maximum value of Long integer) and for 1 million events of each of size 1000 bytes, the Kafka gives the consistent median latency of 1 millisecond even with 50 MB/s throughput. For Pravega, the least is 3 MS and highest is 7 MS with durability writes to bookkeepers.

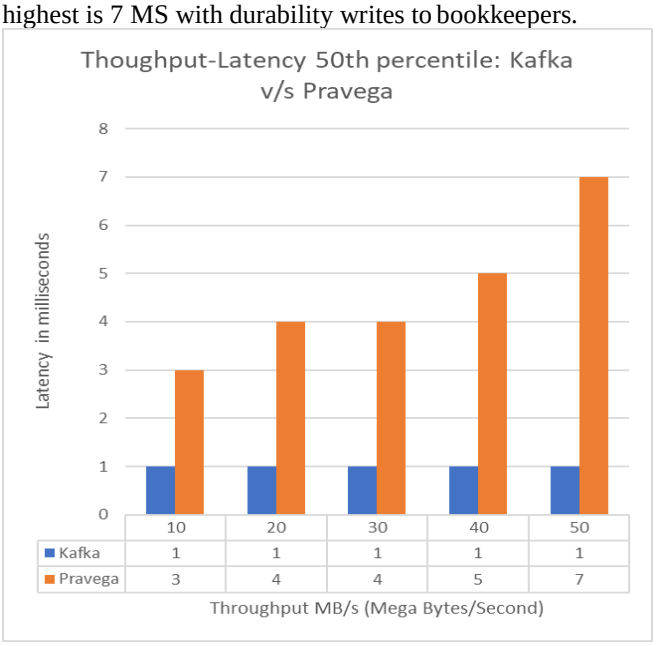

Fig. 10: Median (50th percentile) of Latency vs Throughput

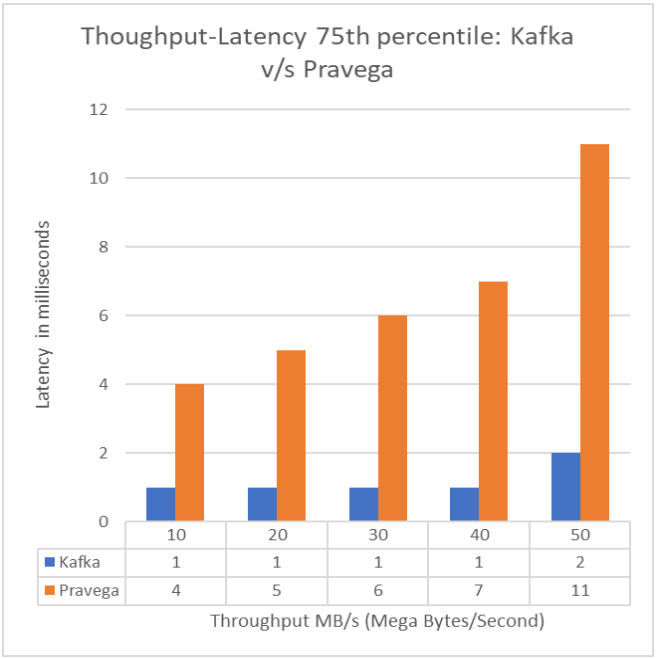

Fig. 11: 75th percentile of Latency vs Throughput

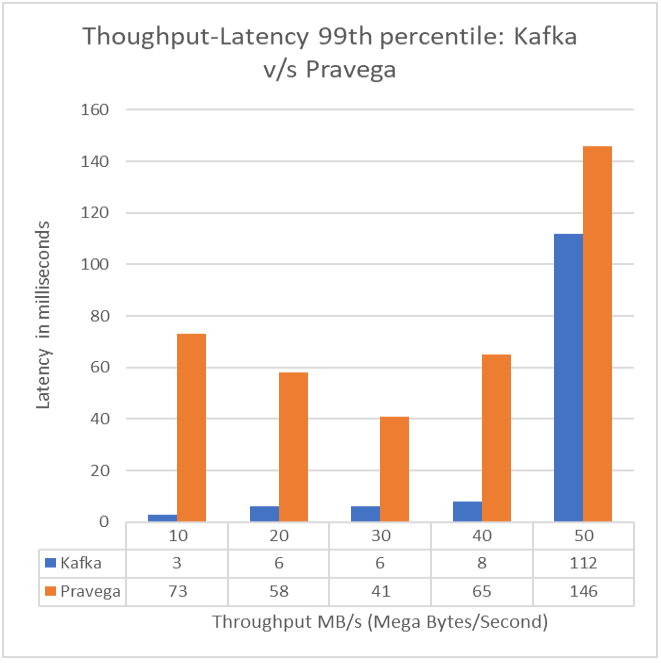

Fig. 12: 99th percentile of Latency vs Throughput

5) Multiple Consumers Benchmarking

Fig. 13. shows the performance difference between 10 consumers of Kafka and Pravega. Note that, the peak performance achieved by Kafka is 1GB/s (Giga Bytes/Second) for approximately 100000 bytes event size. Since Tier 2 reads are involved, Pravega reads are low.

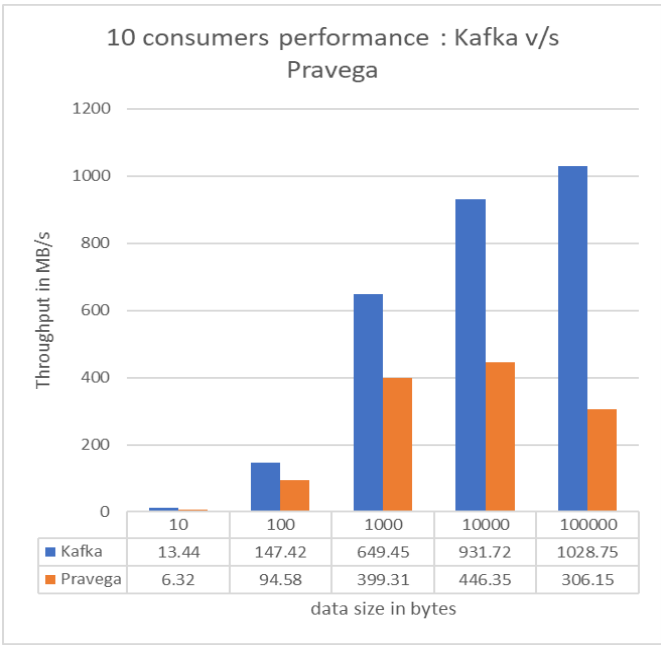

Fig. 13: 10-consumers performance benchmarking: Kafka vs. Pravega

6) End to End Latency Benchmarking

Fig. 14, 15, and 16 shows the median(50th), 75th and 99th latencies comparison between Kafka and Pravega with single producer and single consumer for varying data sizes of 10, 100, 1000, 10000, 100000 bytes. The Kafka records the lower latency than Pravega.

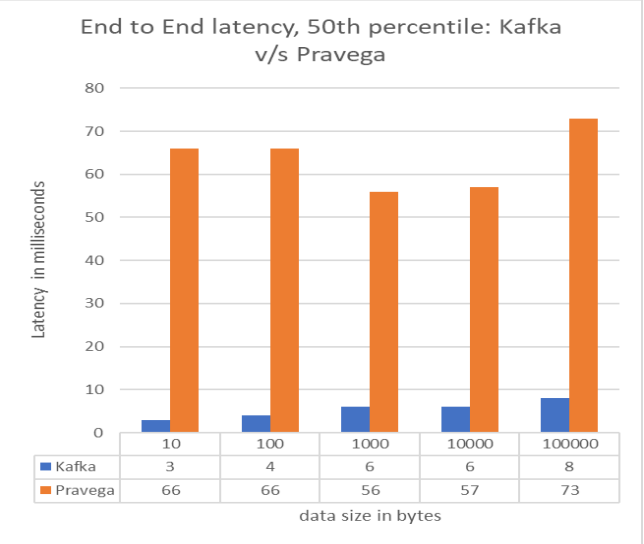

Fig. 14: Median (50th percentile) latency of Kafka vs Pravega

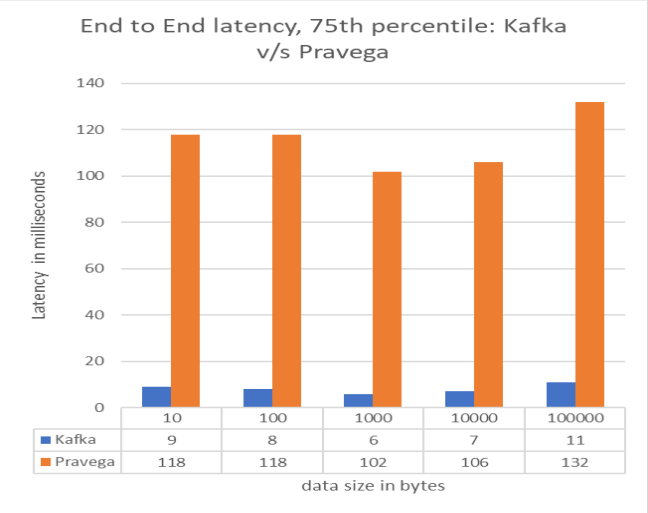

Fig. 15: 75th percentile latency of Kafka vs Pravega

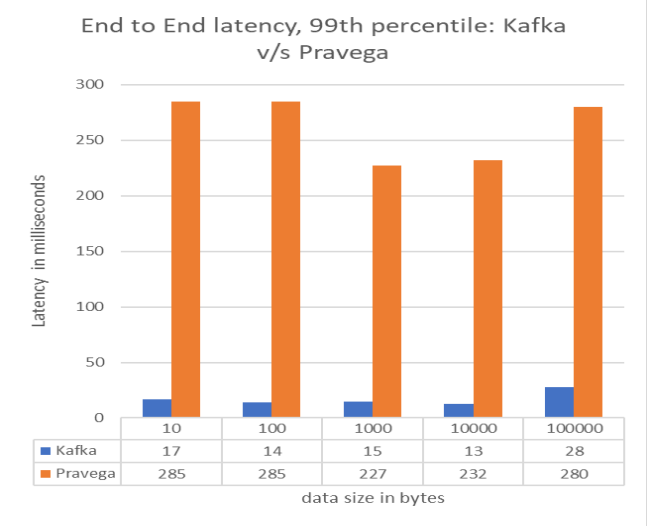

Fig. 16: 99th percentile latency of Kafka vs Pravega

V. CONCLUSION

The design and implementation details of a High-performance benchmarking tool for Kafka and Pravega with multi producers and consumers supported are presented in this paper. This design solves the synchronization issues between multiple producers/consumers and it also identifies the optimal data structures, data serializers, and frameworks to achieve the peak throughput for Kafka and Pravega. The Pravega benchmark tool is compared against Kafka producer and consumer performance benchmark tool and Open messaging benchmark tool too. The design of Pravega benchmark tool, presented in this paper, can be extended to performance benchmarking of any distributed storage systems.

The Pravega streaming storage is optimized for write operations with durability assured. Pravega is younger compared to Kafka and further performance improvements are expected in the future.

ACKNOWLEDGMENT

We thank our Pravega team lead Dr. Flavio Junqueira [16] [19] for his guidance during performance benchmarking. We also thank our other team members, Tom Kaitchuk and Dr. Raul Gracia, for reviewing the source code of the Pravega benchmark tool.

REFERENCES

1. Neha Narkhede, Gwen Shapira and Todd Palino, “Kafka, The Definitive guide, O’reilly series, 1st edition, july 2017.
2. Apache Kafka website : https://kafka.apache.org/ , 2019.
3. Tyler Akidau, Slava Chernyak, and Reuven Lax, “Streaming Systems”, O’reilly series, 1st edition, July 2018.
4. Apache Kafka download: https://kafka.apache.org/downloads#2.3.0 , version 2.3.0, 25th Jun 2019.
5. Apache Kafka source code: https://github.com/apache/kafka , 2019.
6. Pravega website: http://pravega.io , 2019.
7. Pravega source code: https://github.com/pravega/pravega , 2019
8. Pravega Benchmark tool: https://github.com/kmgowda/pravega-benchmark/releases/tag/v1.0
9. Open messaging benchmark tool: https://github.com/openmessaging/openmessaging-benchmark , 2019.
10. Herbert Schildt, Java: The Complete Reference, Oracle Press, 9th Edition, 2014.
11. Brian Goetz, “Java Concurrency in Practice”, Addison-Wesley publications, 9th print, 2010.
12. Java 7/8 , Array blocking queue, https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ArrayBlockingQueue.html , 2019.
13. Java 7/8, Linked blocking queue, https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/LinkedBlockingQueue.html , 2019.
14. Java 7/8, Concurrent Linked queue, https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html , 2019.
15. Maged M. Michael and Michael L. Scot, “Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms”, Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, 1996.
16. Flavio Junqueira and Benjmin Reed, “Zookeeper: Distributed Process Coordination”, O’reilly series, 1st edition, November 2013.
17. Apache Zookeeper , website : https://zookeeper.apache.org/ , 2019.
18. Martin Kleppman, “Desigining Data intensive Applications”, O’reilly series, 1st edition, March 2017.

AUTHORS PROFILE

Mr. Sanjay Kumar NV is an Associate Professor in the Dept. of CSE, KIT, Tiptur, India. He is currently pursuing Ph.D under VTU, Belagavi. He has published several research papers international conference papers since 2009. He is an active member of IETE,ISTE, CSI.

Dr. Keshava Munegowda, is a consultant Engineer at Dell EMC, Bangalore, India. He is an open source developer of Pravega stream-based storage. He holds 5 US patents in the storage domain and has published several research papers in international Journals and Conferences.