Design and Implementation of Storage Benchmark Kit

Keshava Munegowda and N. V. Sanjay Kumar

1 Introduction

The Storage Benchmark Kit (SBK) [1–3] is an open source, vendor neutral, high-performance storage benchmarking software framework. The SBK is containerized using dockers [3], and it is cloud deployable too. It is designed to support any storage device/client with any data type as a payload. The SBK supports multiple writers and readers/callback (push) readers performance benchmarking. The design of SBK is inspired by the Pravega benchmark tool [4, 5]. The Pravega benchmark tool [5] is specific to Pravega [6] and Kafka [7, 8] performance benchmarking, whereas the SBK supports a variety of storage systems such as Apache Bookkeeper [9, 10], Hadoop Distributed File System (HDFS) [11, 12], RabbitMQ [13, 14], RocketMQ [15], NATS [16], NATS Streaming [17], ActiveMQ Artemis [18], NSQ [19], Apache Pulsar [20] along with the existing Pravega and Kafka streaming storage systems.

The SBK also supports the performance benchmarking of database systems such as Apache Derby [21], MySQL [22], PostgreSQL [23], Microsoft SQL [24] and SQLite [25] through Java DataBase Connectivity (JDBC) [26].

The SBK also supports the performance benchmarking of a persistent key-value store such as RocksDB [27] and the distributed key-value store such as FoundationDB [28]. It also supports the performance benchmarking of Protocol buffer-based [29] Record layer [30] of FoundationDB and document-based databases such as FoundationDB Document layer [31] and MongoDB [32]. The performance benchmarking of object storage systems such as MinIO [33] is also supported in SBK. The SBK implements the periodic logging of benchmarking results to the Grafana [34] analytics platform through Prometheus monitoring systems [35].

K. Munegowda (✉)
DellEMC, Bangalore, India

N. V. Sanjay Kumar
KIT, Tiptur, India

N. R. Shetty et al. (eds.), *Emerging Research in Computing, Information, Communication and Applications*, Lecture Notes in Electrical Engineering 790,
https://doi.org/10.1007/978-981-16-1342-5_5

In this paper, the design and implementation of SBK are detailed. To demonstrate the performance benchmarking capabilities of SBK, the XFS file system, HDFS, and Kafka performance benchmarking are conducted. Since Red Hat Enterprise Linux (RHEL) version 7.4 supports XFS as the default file system, we selected XFS for file system benchmarking. In our experiments, it is confirmed that HDFS is stable and high performing in the distributed file systems category and Kafka is stable and performs well in the distributed streaming storage systems category.

2 Design of SBK

The following internal components of SBK are shown in Fig. 1.

2.1 SBK Benchmark

The SBK benchmark parses and processes the application/user supplied or command line arguments, configures the multiple writers, readers, and the component "SBK performance processor." For some of the storage systems/distributed messaging platforms like RabbitMQ [13, 14] and RocketMQ [15], the SBK initiates the callback (push) asynchronous readers too.

2.2 Writers and Readers/Callback (Push) Asynchronous Readers

These components initiate the performance benchmarking of write and read operations. These components implement Burst Mode/Max Throughput Mode [4] to measure the maximum write/read throughput of a storage system, Throughput Mode [4] and Rate Limiter Mode [4] to analyze the latency variations under controlled throughput or rate of events/records and End to End Latency Mode [4] to determine the total time duration between time to write a data record and reading the same data record.

2.3 Data Type Handler

The Data Type handler defines the type of data and methods/operations to operate on the data. Example data type handlers are Byte Array [36], Java NIO Byte Buffer [36], Java String [36], and Protocol buffers [29].

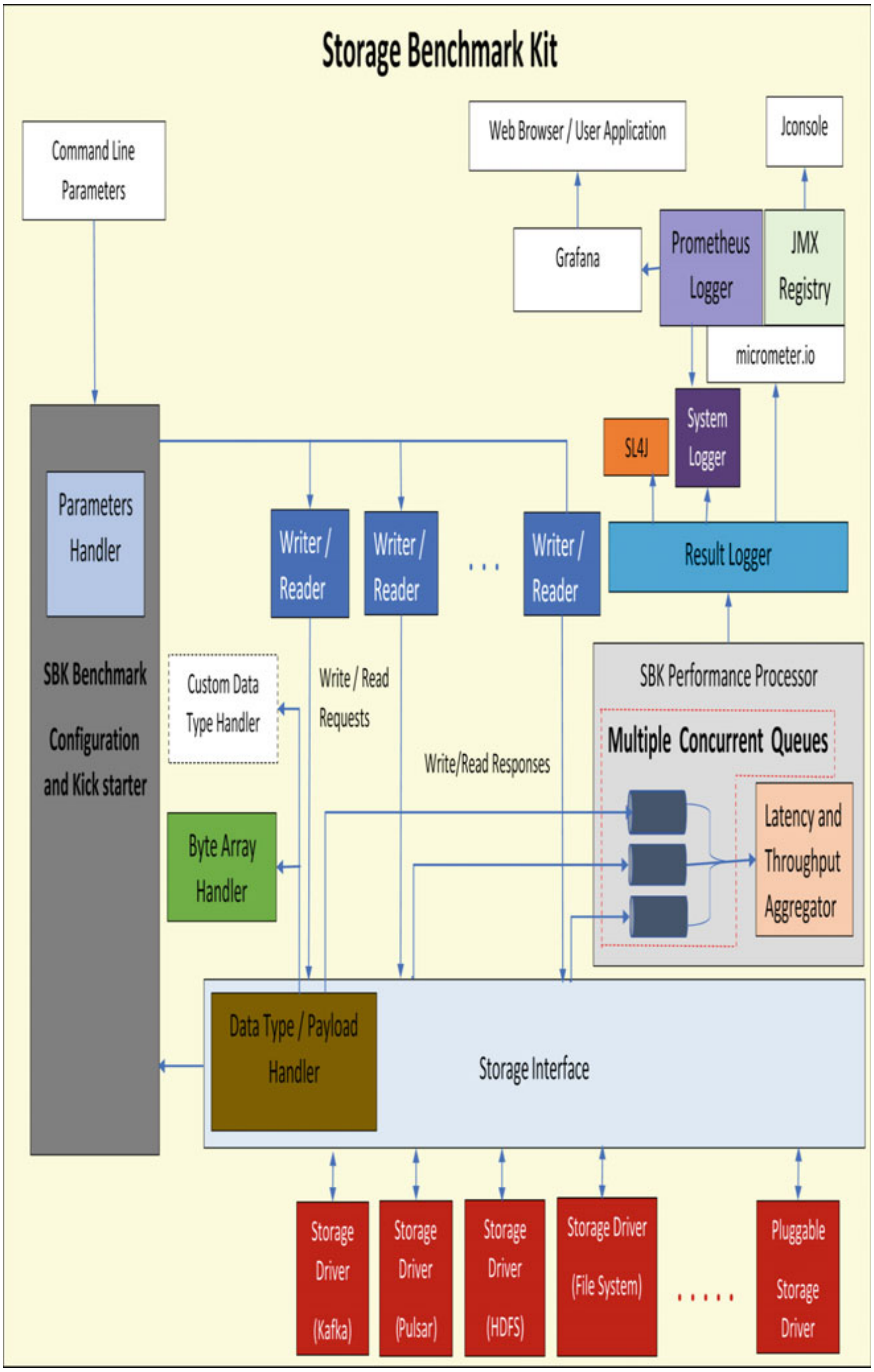

Fig. 1 Design of SBK

2.4 *Storage Interface and Driver*

The SBK defines and implements default methods for the storage interface which are extended and used to implement a custom and pluggable storage driver for any storage device/client. This pluggable storage driver component defines the write and read operations of the storage device/client. A single storage driver component implements a single or multiple instances storage device/client. The pluggable storage driver either chooses one of the available data type handlers or defines a new custom data type handler.

2.5 *SBK Performance Processor*

The SBK Performance Processor solves the synchronization issues between the multiple writers, readers and the response threads which are created upon completion of asynchronous write or read operations. This component uses multiple concurrent queues [37–39] to store an information record enclosing performance values such as start time, end time, number of records, and number of bytes in the total number of records for a single or multiple write/read operations. The java concurrent linked queue [38, 39] provides a thread-safe and wait-free/non-blocking [37] application programming interface (APIs) for enqueue and dequeue operations. Multiple response threads of the write/read completion or writers and readers enqueue the performance values to these multiple concurrent linked queues, but a single dedicated thread named "Latency and Throughput Aggregator" deques the stored performance values from these multiple concurrent linked queues to calculate the latency and throughput values. The SBK treats the response time of write/read as the latency value. The latency counts are stored in an array in which latency value is used as an index. These latency counts are extracted for calculation of the latency percentiles [4]. The latency percentiles calculation method used in SBK is inspired by counting-sort algorithm with time complexity of Big O (Maximum Latency).

2.6 *Result Logger*

This component receives the benchmark results such as throughput values, average, and maximum latency values and latency percentiles for every predetermined time interval from the SBK performance processor component. These benchmark results are logged to a local output device. The SBK uses the micrometer [40] software interface to log the results to the Prometheus [35] monitoring system and SL4J (Simple Logging façade For Java) [41] logging system. The Grafana [34] analytics platform receives these benchmark results from Prometheus [35] monitoring system.

3 Implementation Details

The SBK is implemented in Java 8 and is open sourced in the Git repository hub [1]. The SBK Docker images are available at the Docker Hub [3] too. The SBK Release Version 0.8 [2] is used for performance benchmarking of the file system, HDFS and Kafka presented in this paper. The SBK GitHub [1] details the guidelines for open source developers to improve the SBK source code and to add a new driver for performance benchmarking of any other storage system.

4 Results and Discussion

4.1 File System Performance Benchmarking

The Flexible IO (FIO) [42] and IOzone [43, 44] are the commonly used file system benchmarking tools. These tools are implemented in the programming language C. The SBK uses the Java file channel [36] APIs and ByteBuffer [36] as data record for the file write and read operations. The hardware and software configurations used for performance benchmarking are listed in Table 1.

Single writer File System Performance Benchmarking: Fig. 2 shows the Grafana snapshot of the single writer file system performance in terms of MB/s (Mega Bytes/second). In our benchmarking experiments, we set the record size as 1,000,000 bytes (approximately 1MB) and the total file size to write is 1 TB (Tera

Table 1 Hardware and software configuration of the test setup

Components	Remarks
Number of compute nodes	4 Nodes 1 for HDFS client/Kafka client 1 XFS File system node/HDFS Name Node 3 for Kafka brokers/HDFS Data nodes
CPUs (Central Processing Unit) per compute node	32 CPUs. Each of CPU is 64 Bit, 2.6 GHz (Giga Hertz)
RAM (Random Access Memory) per node	350 GB (Giga Bytes)
Hard Disk per node	HDD (Hard Disk Drive) of size 3 TB (Tera Bytes)
Ethernet per node	10 Gbps (Giga Bits/second) Network
Operating System	RHEL (RedHat Enterprise Linux) Release version 7.4

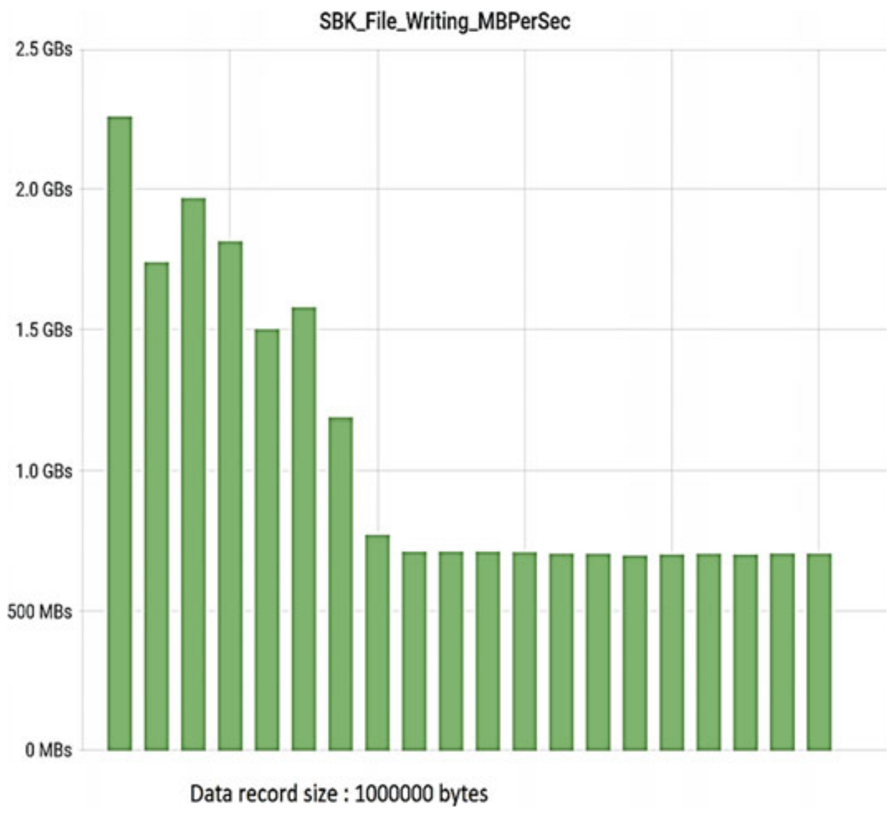

Fig. 2 Single File Writer throughput performance in MB/s

Bytes). Maximum throughput of 2.2 GB/s and average write throughput range of 750–780 MB/s are observed in our test setup.

Single and Multiple Readers File System Performance Benchmarking: Fig. 3 shows the read throughput of the single reader with a record size of 1,000,000 bytes (approximately 1MB) to read a 1 TB size file. The peak throughput is 4–5 GB/s and an average read throughput is 650–680 MB/s.

The SBK scales high with multiple readers. Figure 4 shows the read throughput of 10 file readers, the peak performance of 40–50 GB/s (Giga Bytes/second), and an average performance of 6–6.4 GB/s are observed and it indicates that the SBK scales high with multiple readers.

4.2 Hadoop File System (HDFS) Performance Benchmarking

TestDFSIO (Distributed File System Input-Output) [12, 45] is the de-facto performance benchmark tool for HDFS. This tool uses the Map-Reduce framework/programming model and hence achieves more parallelism for HDFS write and read operations. But, SBK uses the HDFS Java stream APIs rather than Map-Reduce framework. Thus, the SBK does the performance benchmarking of the raw

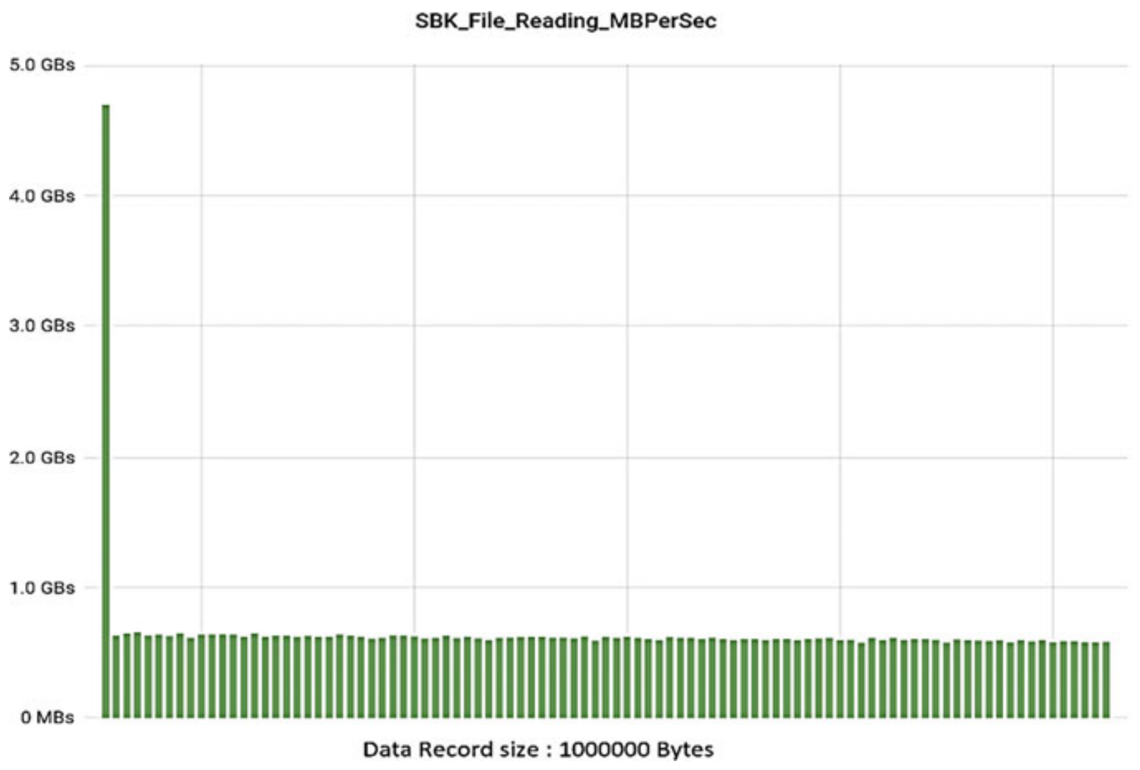

Fig. 3 Single File Reader throughput performance in MB/s

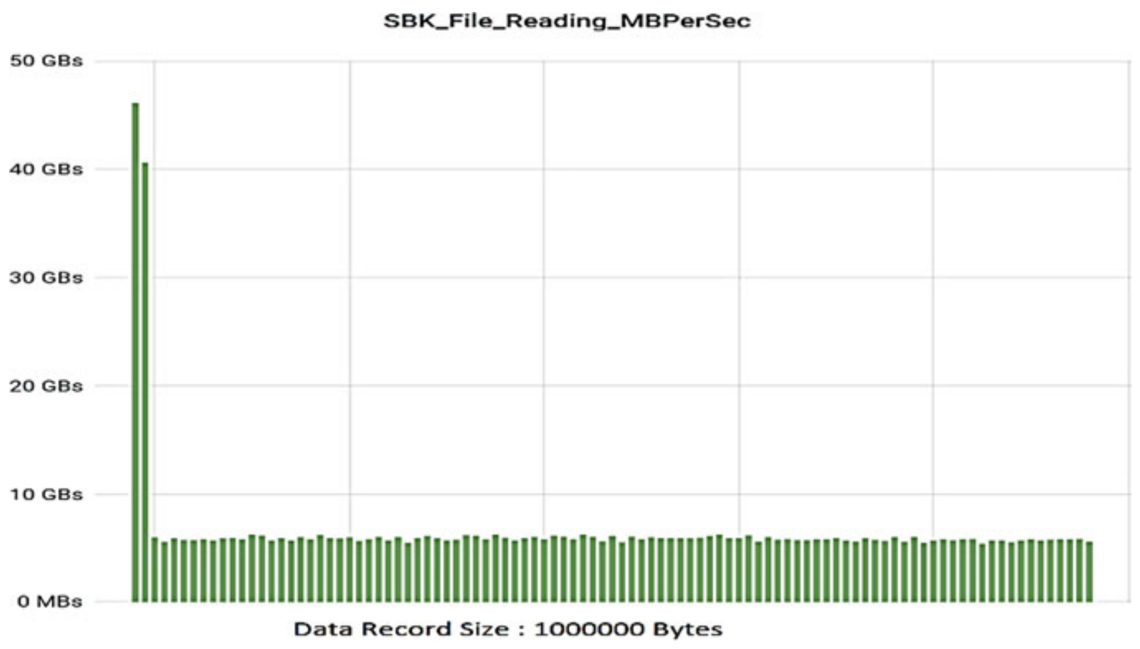

Fig. 4 10 File Readers throughput performance in MB/s

distributed file system with Byte Array as data type. The Hadoop version 3.2.0 [45] is used for performance benchmarking.

Single writer HDFS performance benchmarking: Like any other file systems, the HDFS supports write, append and overwrite of a file by a single writer only.

Figure 5 shows the Grafana graph snapshot of the write throughput of the single writer of one file. Note that, the max throughput is in the range of 500–550 MB/s (Mega Bytes per Second), and the average throughput range is 200–280 MB/s. In our benchmarking experiments, we set the block size as 1,000,000 bytes (approximately 1 MB) and the total file size to write is 1 TB (Tera Bytes).

Single and Multiple HDFS Readers performance benchmarking: The SBK scales high with multiple readers. Figure 6 shows the read throughput of the single reader and 10 readers of each reader of block size 1,000,000 bytes (approximately 1MB) to read 1 TB size file. Note that, with 10 readers SBK records the peak throughput of more than 2 GB/s (Giga Bytes per second).

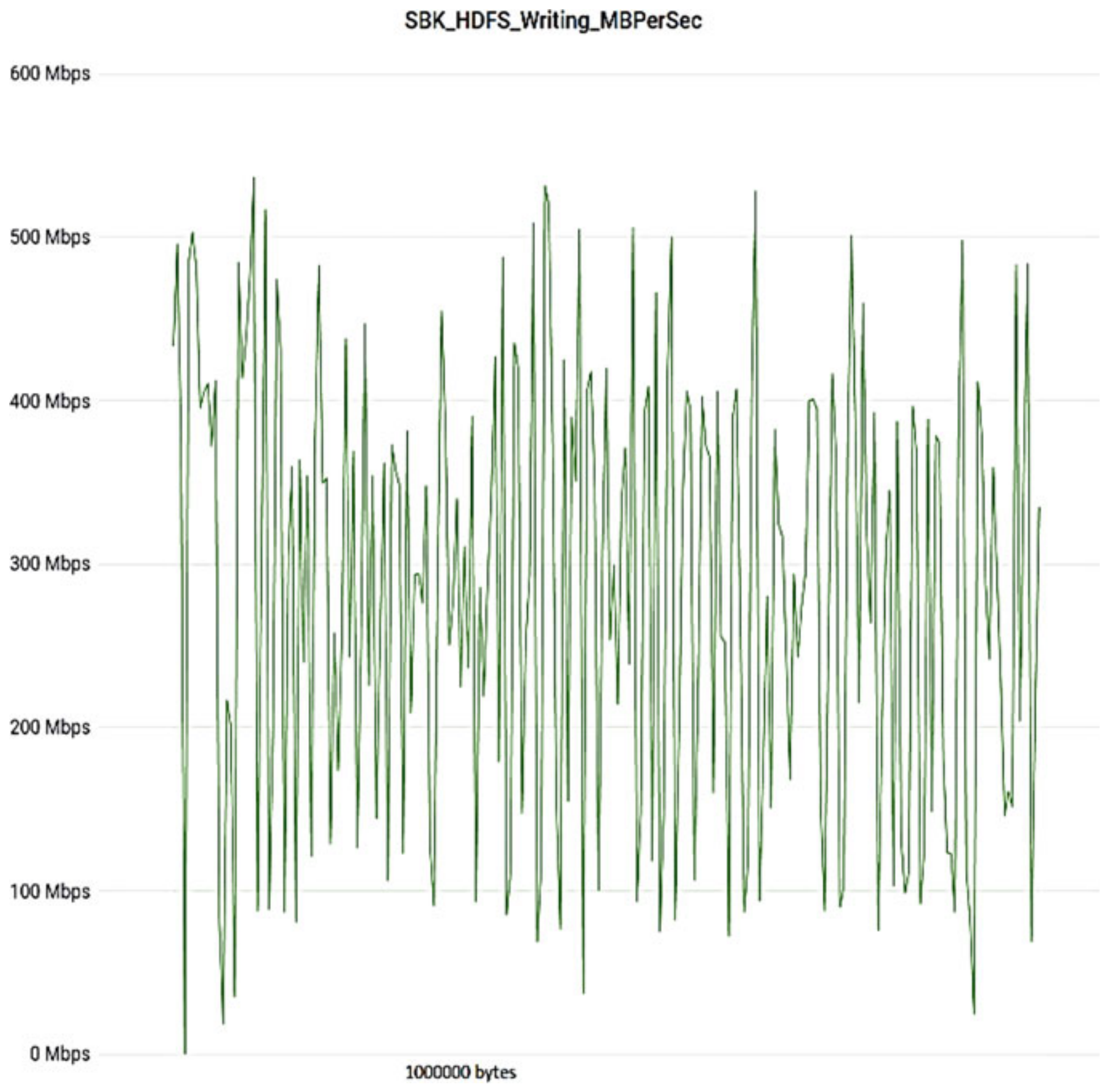

Fig. 5 Single HDFS Writer throughput performance in MB/s

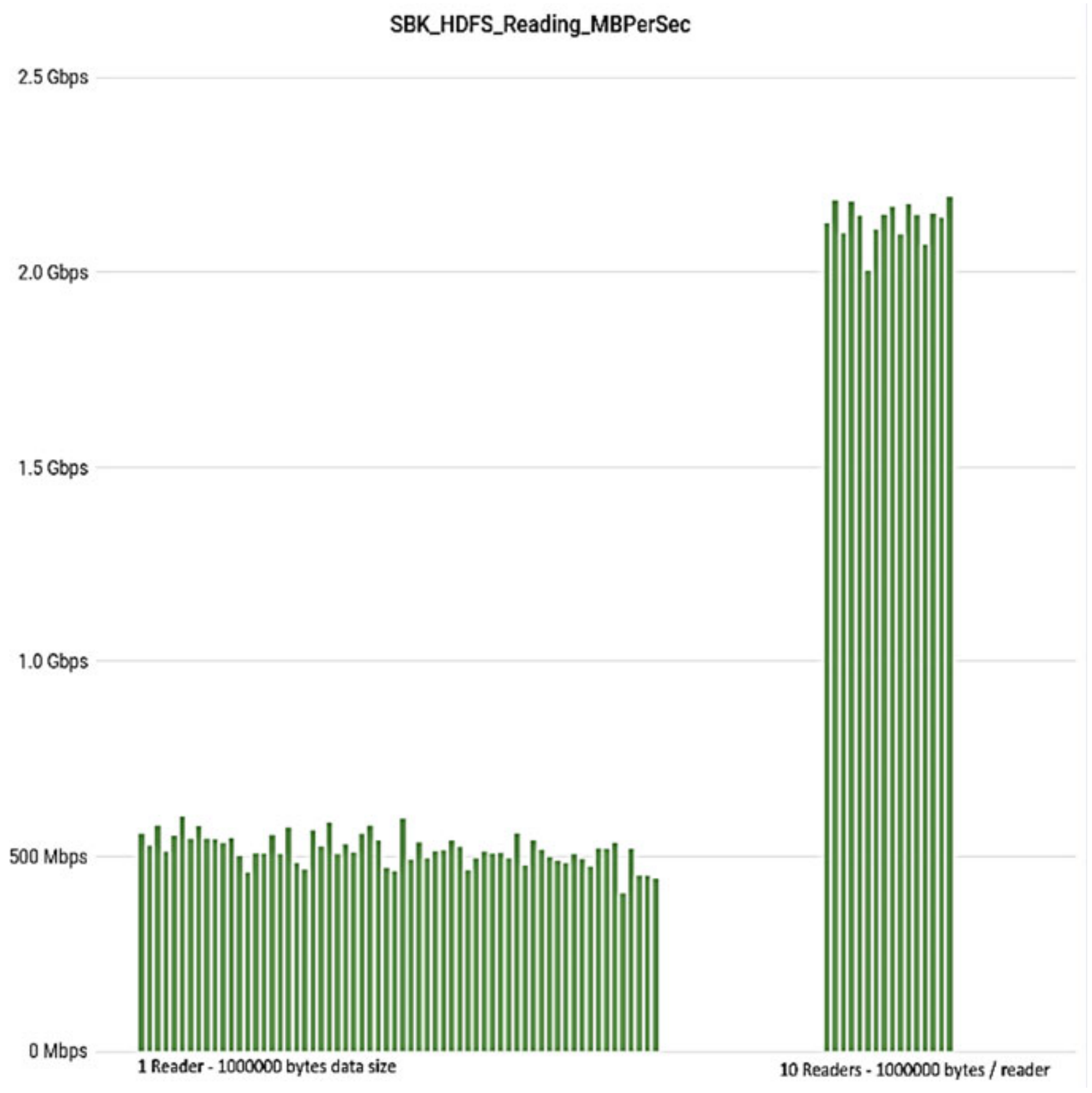

Fig. 6 Single and 10 HDFS Readers throughput performance in MB/s

4.3 Kafka Performance Benchmarking

The Kafka release version 2.4.1 [46] is used for performance benchmarking. A Kafka topic with 15 partitions, 3 replicas in which 2 are in-sync replicas, with the hardware configuration shown in Table 1 is used for performance benchmarking. The max poll (maximum poll records for consumer) configuration is set to 32-bit integer max value to read the maximum available records from the Kafka broker. Assigning the higher value for max.poll configuration value improves the Kafka read performance. The Byte Array [36] is used for data serialization and deserialization.

An existing Kafka (version 2.4.1) producer benchmark tool [4, 8, 46] for write performance benchmarking and consumer benchmark tool [4, 8, 46] for read performance benchmarking, are limited only to a single producer/writer and

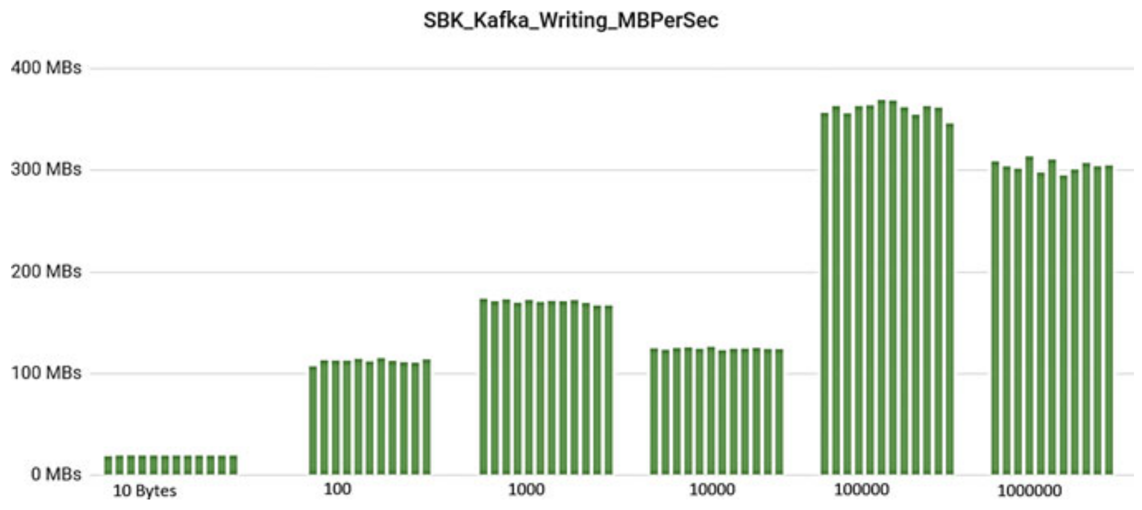

Fig. 7 Kafka Single producer throughput performance in terms of MB/s

consumer/reader, respectively, whereas SBK supports benchmarking of Kafka write and read with multiple writers and readers.

The open messaging benchmark [47, 48] software supports the Kafka performance benchmarking. As of today, the open messaging benchmarking is developed with the concept of benchmarking distributed messaging and streaming platforms, whereas SBK adheres to the concept of benchmarking any storage system with any type of data payload. The SBK supports the benchmarking of distributed messaging and streaming platforms along with any generic persistent or non-persistent storage platform.

Single Kafka Producer Benchmarking: The Kafka performance benchmarking is conducted with varying data sizes from 10 bytes to 100,000 bytes. The Grafana results snapshot of throughput variations for these different data sizes for a single producer is shown in Fig. 7.

Note that, for a single producer, the SBK records the peak performance in the range of 280–370 MB/s for approximate data sizes of 100 K and 1 MB. The same throughput results can be depicted in terms of records/second or writes/second (wps) as shown in Fig. 8. For smaller record/event sizes, the wps values are high and for larger data size the wps values are very low. The peak throughput is recorded in the range of 1.5–1.8 million wps. Due to a large gap between 1.8 million wps to 1000 wps, the lower wps values for data size of 1MB are not clearly visible in Fig. 8.

In our experiments, we observed that, reducing the value for the configuration parameter log.flush.interval.messages improves the durability of writing data by flushing data to the storage disk/device more frequently. But, it reduces performance. Hence, in our test setup, we have set this value to 64-bit long integer maximum value as default. Since the Kafka write operations are asynchronous, Fig. 9 shows the latency values of the write operations of the single producer. In our experiments, we observed that the median write percentile of 5 ms (milliseconds) for 10 bytes of data size and 60 ms for 1MB data size with the maximum possible throughput.

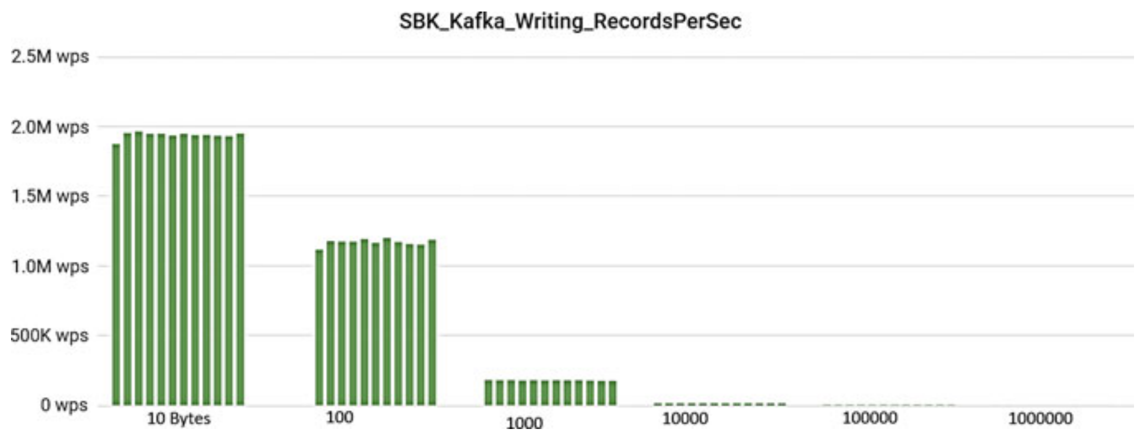

Fig. 8 Kafka single producer throughput performance in terms of Records/Second or Writes/Second (wps)

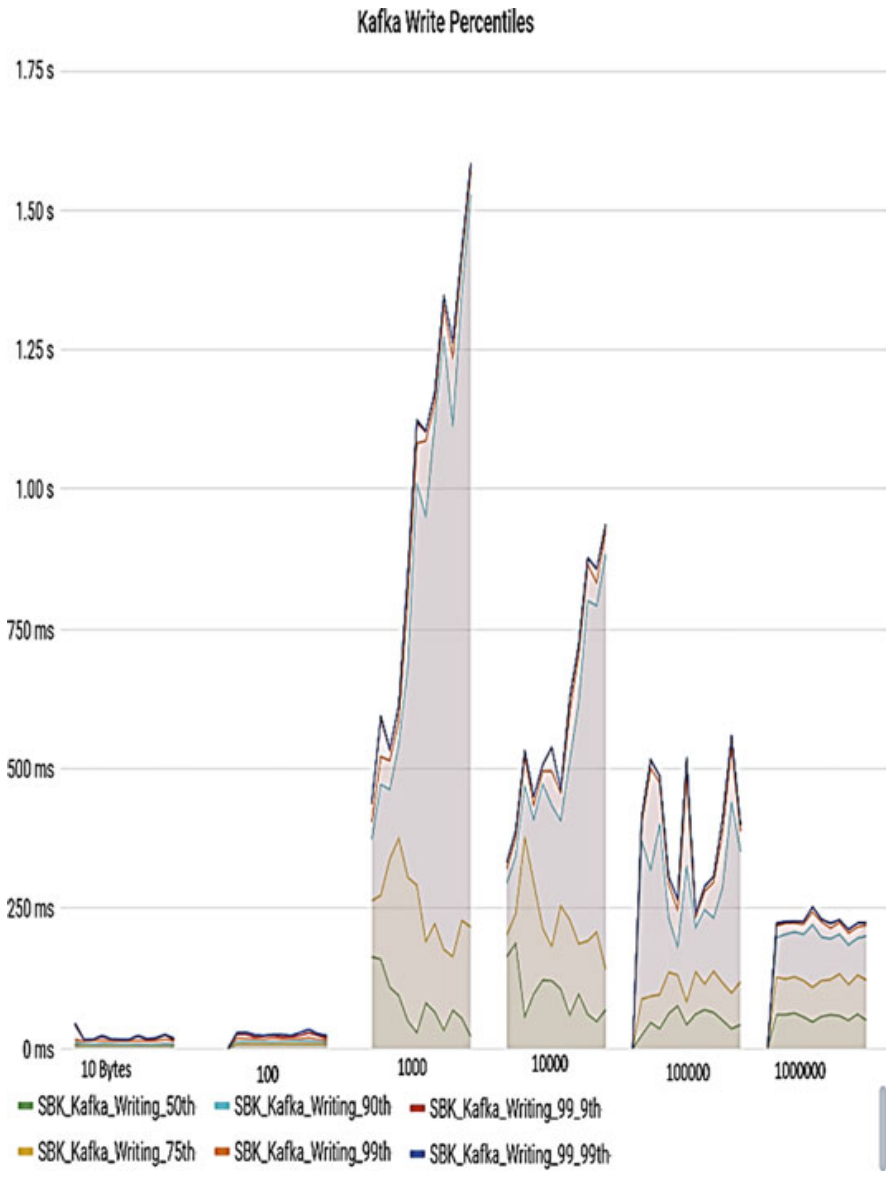

Fig. 9 Single Kafka Producer latency Percentiles

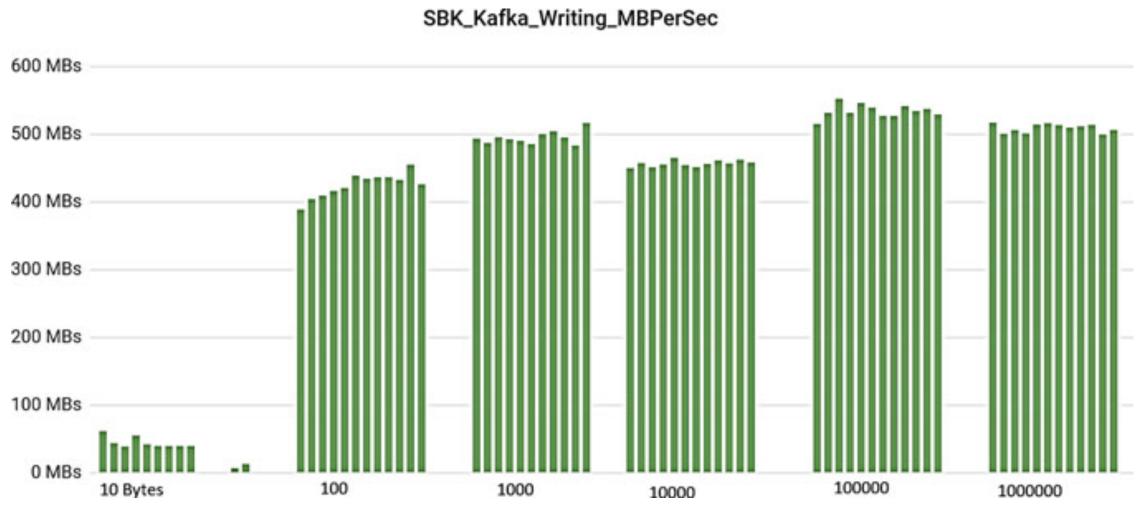

Fig. 10 Kafka 10 producers throughput performance in terms of MB/s

Multiple Kafka Producers Benchmarking: The SBK scales high with multiple writers/producers. Figure 10 shows the performance benchmarking of 10 Kafka producers. Note that, the SBK achieves the maximum throughput range of 500–550 MB/s for record (data) sizes of 100 K to 1 MB. The same throughput results can be depicted in terms of records/second or writes/second as shown in Fig. 11. Note that, for smaller data sizes less than 100 bytes, the SBK records the peak throughput of more than 4 million wps.

Single Kafka Consumer benchmarking: Fig. 12 shows a single Kafka consumer performance for the data sizes of 10, 100, 1000, 10000, 100000, and 1000000 bytes. The SBK records the maximum throughput range of 35–45 MB/s for 10 bytes data record size. The throughput range of 400–450 MB/s is observed for data sizes of 10 K to 1 MB. Figure 13 shows the single Kafka consumer throughput performance in terms of Records/Second or Reads/Second (rps). In our experiments, we have observed that by decreasing the Kafka reader configuration parameter max.poll drastically reduces the read performance. Hence, in our test setup, the max poll is set to 32 Bit Max integer value.

Multiple Consumers benchmarking: Fig. 14 shows the 10 consumers performance benchmarking for the data sizes of 10, 100, 1000, 10000, 100000, and 1000000 bytes. It shows that the SBK records the peak performance range of 1.5–1.8 GB/s (Giga Bytes/second) for the data sizes of 100 K to 1 MB. Figure 15 shows the same performance benchmarking in terms of Records/Second or Reads/Second.

End to End Latency Benchmarking: Fig. 16 shows the End to End latency of the Kafka single writer and reader for 10, 100, 1000, 10000, 100000, and 1000000 bytes of data sizes. In our experiments of End to End latency mode, the SBK is set to flush every record and hence it attempts to record the least latency at a low throughput rate. For the data size of 100K or less, the median (50^{th} percentile) latency is in the range of 1–3 ms. For the data size of 1MB, the median latency is in the range of 10–15 ms.

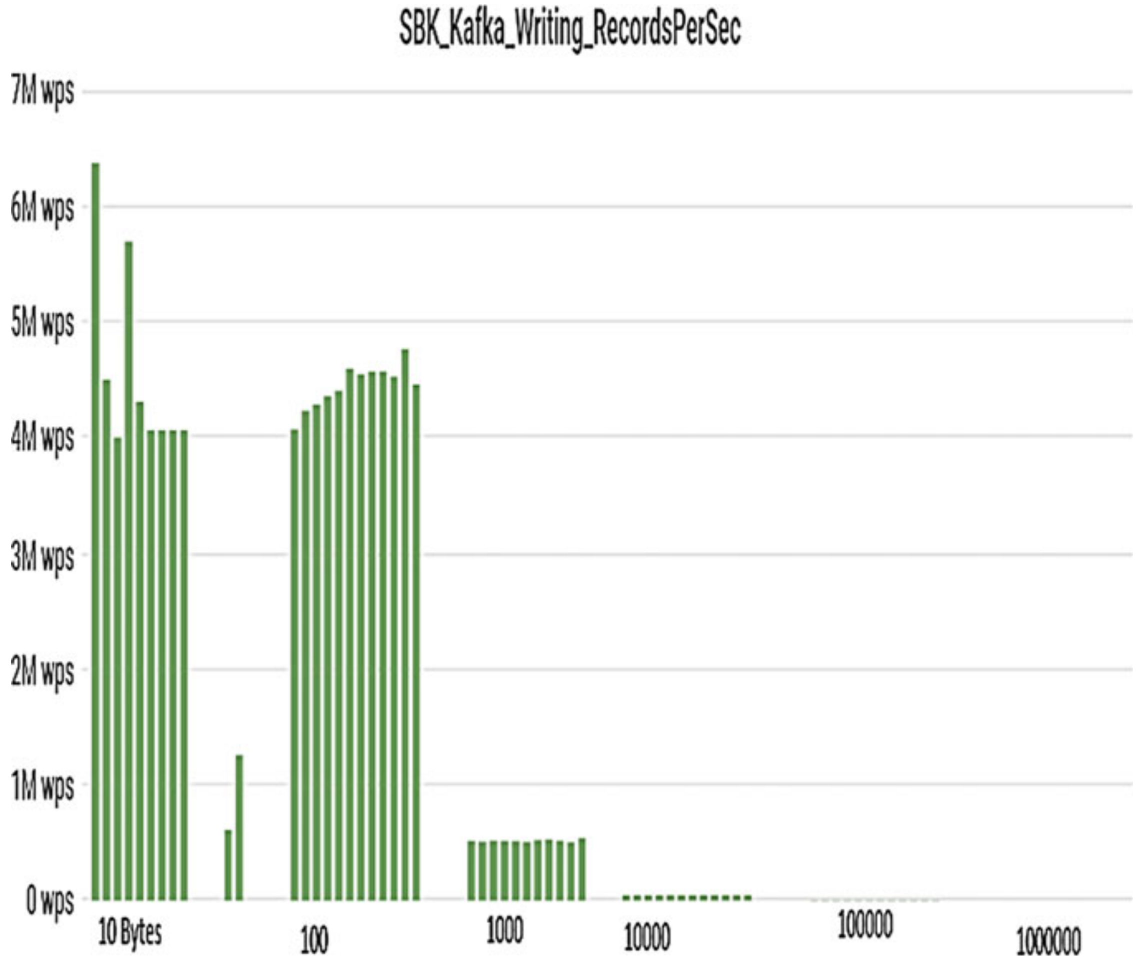

Fig. 11 Kafka 10 producers throughput performance in terms of Records/Second or Writes/Second (wps)

5 Conclusion

The design and implementation of the SBK framework, presented in this paper, scales high with multiple readers and writers. This design solves the synchronization issues between multiple writers, readers/callback (push) readers, and it also describes the best suited data structures such as multiple concurrent queues to measure the maximum throughput and low latency for any storage device/client/cluster. The design of SBK exports the standard storage interface APIs which can be extended to include a storage driver to conduct the performance benchmarking of any custom storage device/client. The SBK currently supports benchmarking of a wide category of storage systems such as local mounted file systems, distributed file systems, distributed messaging, streaming storage platforms, key-value storage systems, database systems, and object storage systems. SBK can be used as a common framework to conduct performance benchmarking among similar category storage systems. For example, different database systems such as MySQL, Apache Derby, PostgreSQL, FoundationDB document layer, and MongoDB can be compared with respect to performance using SBK. Different streaming storages such as Kafka and

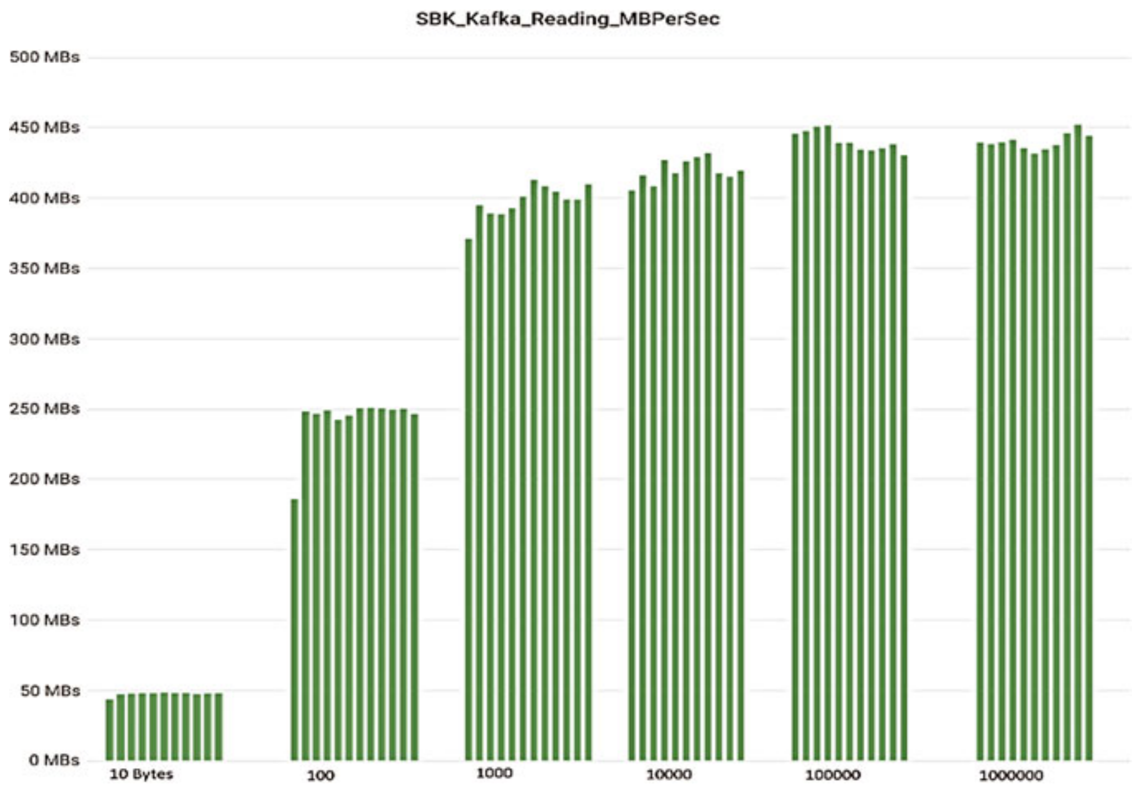

Fig. 12 Single Kafka consumer throughput performance in terms of MB/s

Pulsar can be compared using SBK. The SBK can be used to compare the performance of different file systems too. The design of SBK is flexible to enclose the performance benchmarking of non-persistent in-memory message queues also.

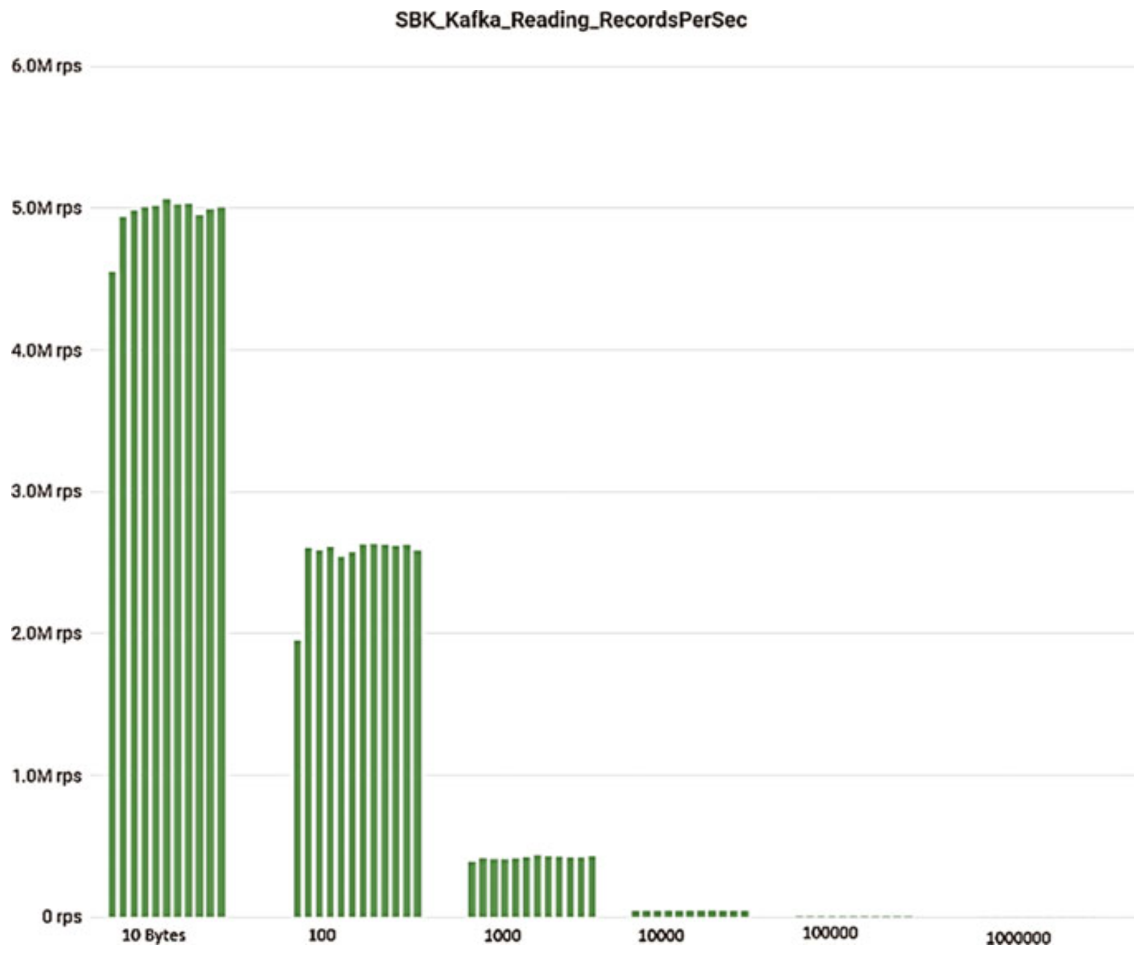

Fig. 13 Single Kafka consumer throughput performance in terms of Records/Second or Reads/Second (rps)

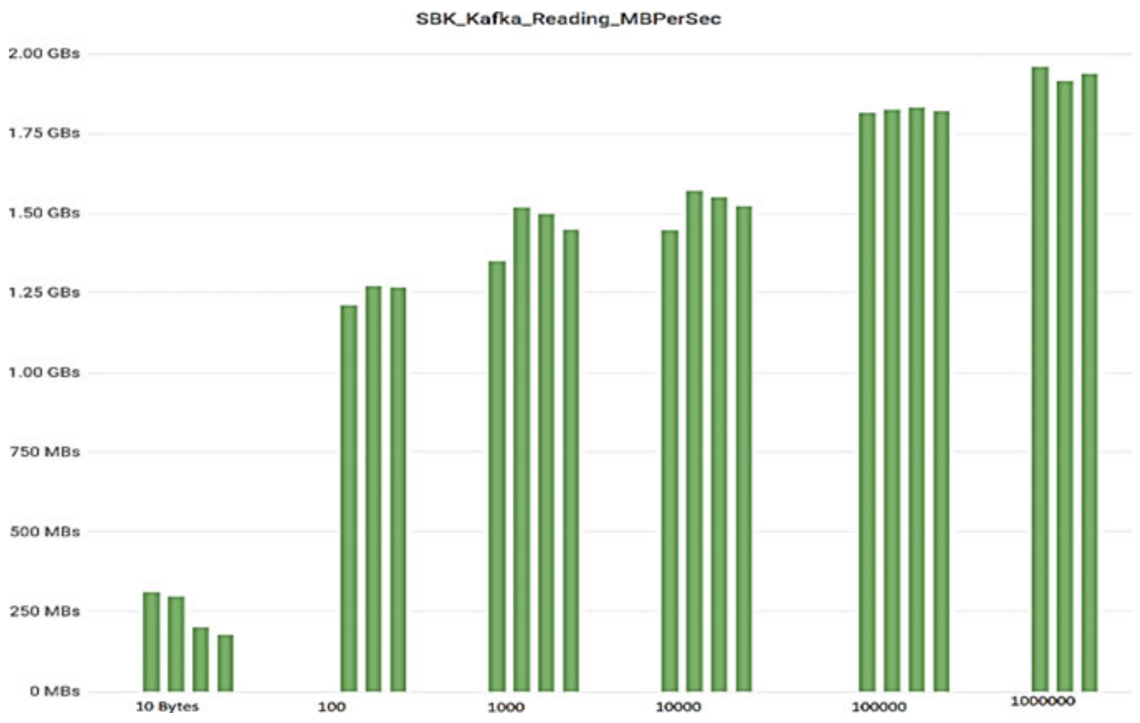

Fig. 14 Kafka 10 Consumers throughput performance in terms of MB/S

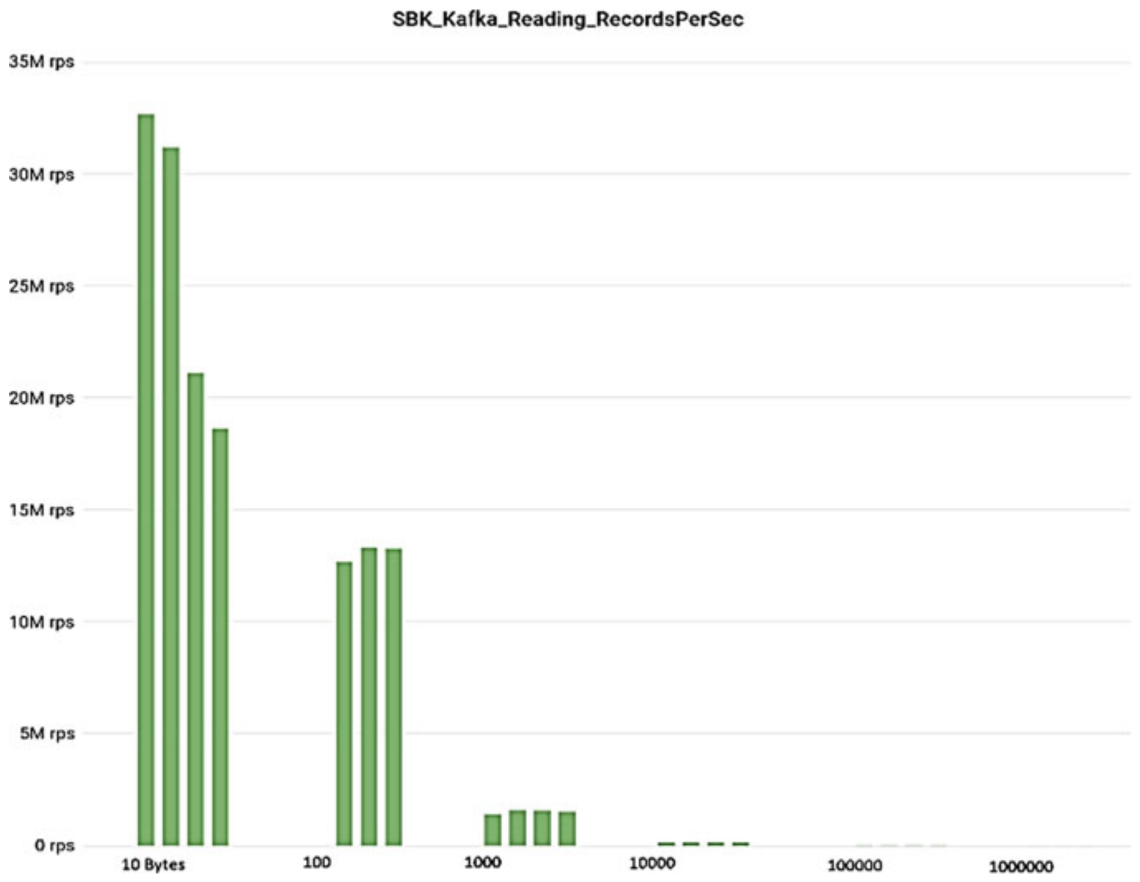

Fig. 15 Kafka 10 Consumers throughput performance in terms of Records/Second or Reads/Second (rps)

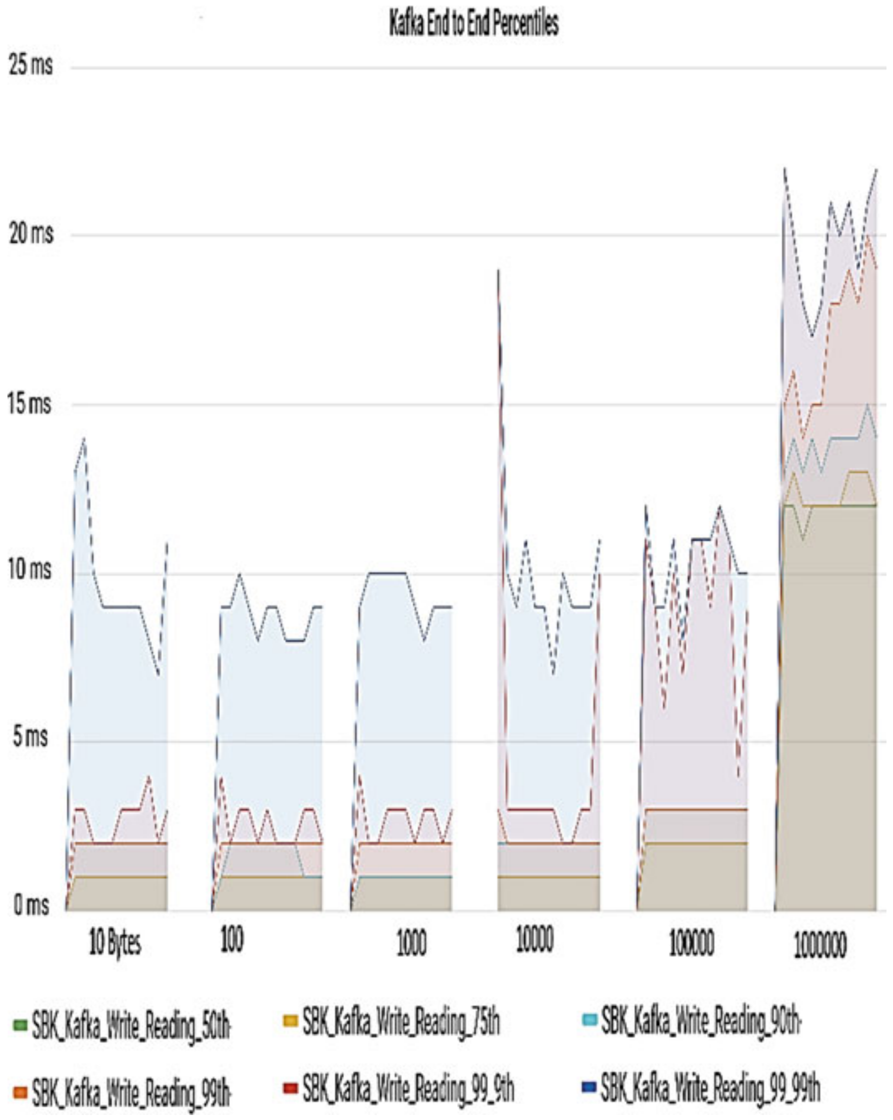

Fig. 16 Kafka End to End Latency percentiles

References

1. Storage Benchmark Kit (SBK). https://github.com/kmgowda/SBK (2020)
2. SBK Releases. https://github.com/kmgowda/SBK/releases (2020)
3. SBK docker images. https://hub.docker.com/repository/docker/kmgowda/sbk (2020)
4. Sanjay Kumar NV, Munegowda K (2019) Distributed streaming storage performance benchmarking: Kafka and Pravega. Int J Innov Technol Exploring Eng (IJITEE) 9(2S). ISSN: 2278-3075
5. Pravega Benchmark Tool, Releases. https://github.com/kmgowda/pravega-benchmark/releases (2020)
6. Pravega website. http://pravega.io (2020)
7. Narkhede N, Shapira G, Palino T (2017) Kafka, The Definitive Guide, O'reilly series, 1st edn
8. Apache Kafka website. https://kafka.apache.org/ (2020)
9. Junqueira F, Kelly I, Reed B (2013) Durability with Bookkeeper. In: ACM SIGOPS operating systems review, vol 47, issue 1, January 2013
10. Apache Bookkeeper website. https://bookkeeper.apache.org/ (2020)
11. White T (2015) Hadoop definitive Guide,O'reilly series, 4th edn

12. Apache Hadoop website. https://hadoop.apache.org/ (2020)
13. Roy G (2018) Rabbit MQ in Depth. Manning Publications
14. RabbitMQ website. https://www.rabbitmq.com/ (2020)
15. RocketMQ website. https://rocketmq.apache.org/ (2020)
16. NATS 2.0 website. https://nats.io/ (2020)
17. NATS Streaming. https://nats.io/download/nats-io/nats-streaming-server/ (2020)
18. Apache ActiveMQ Artemis. https://activemq.apache.org/components/artemis/ (2020)
19. NSQ website. https://nsq.io/ (2020)
20. Apache Pulsar website. http://pulsar.apache.org/ (2020)
21. Apache Derby website. https://db.apache.org/derby/ (2020)
22. MySQL website. https://www.mysql.com/ (2020)
23. PostgreSQL website. https://www.postgresql.org/ (2020)
24. Microsoft SQL website. https://www.microsoft.com/en-in/sql-server/sql-server-2019 (2020)
25. SQLite website. https://www.sqlite.org/index.html (2020)
26. Deitel and Deitel (2015) Java How to Program, 10th edn. Pearson Publications
27. RocksDB. website https://rocksdb.org/ (2020)
28. FoundationDB, Website. https://www.foundationdb.org/ (2020)
29. Google Protocol Buffers. Website https://developers.google.com/protocol-buffers (2020)
30. Chrysafis C, Collins B, Dugas S, Dunkelberger J, Ehsan M, Gray S, Grieser A, Herrnstadt O, Lev-Ari K, Lin T, McMahon M, Schiefer N, Shraer A (2019) FoundationDB record layer: a multi-tenant structured datastore. In: SIGMOD '19: Proceedings of the 2019 international conference on management of data, pp 1787–1802
31. FoundationDB Document layer. website: https://foundationdb.github.io/fdb-document-layer/ (2020)
32. MogoDB. website: https://www.mongodb.com/ (2020)
33. MinIO. website: https://min.io/ (2020)
34. Grafana Website. https://grafana.com/ (2020)
35. Prometheus Website. https://prometheus.io/ (2020)
36. Schildt H (2014) Java: the complete reference, 9th edn. Oracle Press
37. Michael MM, Scot ML (1996) Simple, fast, and practical non-blocking and blocking concurrent queue algorithms In: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing
38. Goetz B (2010) Java concurrency in practice. Addison-Wesley publications, 9th print
39. Java 7/8, Concurrent Linked queue. https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html (2020)
40. Micrometer IO website https://micrometer.io/ (2020)
41. SL4J website: http://www.slf4j.org/ (2020)
42. FIO, website: https://github.com/axboe/fio (2020)
43. Munegowda K, Raju GT, Poddar S (2012) FFSB and IOzone: file system benchmarking tools, features and internals. In: Proceedings of Embedded Linux Conference (ELC), Barcelona, Spain, Europe
44. IOzone website: http://www.iozone.org/ (2020)
45. Hadoop version 3.2.0 release: https://hadoop.apache.org/release/3.2.0.html (2019)
46. Kafka version 2.4.1 release: https://kafka.apache.org/downloads (2020)
47. Open messaging framework. website: http://openmessaging.cloud/docs/benchmarks/ (2020)
48. Open messaging source code. https://github.com/openmessaging/openmessaging-benchmark (2020)