SBP: Storage Benchmark Protocol

Keshava Munegowda
Vice President, SecDB Engineering *Goldman Sachs*
Bengaluru, India
keshava.gowda@gmail.com

2022 4th International Conference on Circuits, Control, Communication and Computing (I4C) | 979-8-3503-9747-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/I4C57141.2022.10057795

***Abstract*—The Storage Benchmark Protocol (SBP) is defined, designed and implementation details are discussed in this research paper. The SBP is used for an optimal performance benchmarking of high performance distributed big data storage systems. The SBP defines a network communication format and methods to consolidate the performance benchmarking parameters from single/multiple storage performance benchmarking entities/nodes. The SBM (Storage Benchmark Monitor) implements the SBP as a server. The SBP client is implemented as part of the SBK (Storage Benchmark Kit) framework.**

Keywords—Benchmarking, Big Data, File Systems, GRPC, Latency, Performance, Percentile, PerL, Protocol Buffers, Quartile, SBK, SBM, SLC, SBP, Storage, Throughput.

I. INTRODUCTION

The SBK (Storage Benchmark Kit)[1][2][3] is a high performance and scalable open-source software framework to measure performance of any type of storage system with any type of data/payload and any time stamp such as milliseconds, microseconds and nanoseconds. The SBK supports performance benchmarking with multiple readers and writers too. In any distributed big data storage systems, multiple instances of SBKs can be executed in single/multiple systems/nodes to load the heavy data to the storage system and conduct the performance analytics. In such situations , each SBK instance measures the number of bytes read/written , number of records read/written and large number of millions and billions of latency values and then it calculates the performance result metrics such as throughput , quartile and percentile of latency values. Manually analyzing the performance values and result metrics from every SBK instance is cumbersome and error-prone activity. Thus, this research paper introduces SBP (Storage Benchmark Protocol).

II. STORAGE BENCHMARK PROTOCOL

The SBP defines the message/data formats to be exchanged between a single server and multiple clients. Each SBP client sends the performance values containing the number of bytes read and written , number of records read and written and the series of latency values to the SBP server. The SBP server consolidates all these performance results and calculates the throughput and percentiles of latency values and SLC (Sliding Latency Coverage) factors[4]. In the SBK framework [2][3] and as part of this paper, the SBP Server is entitled as SBM (Storage Benchmark Monitor).

Figure 1 shows the distributed cluster deployment of SBK and SBM. The SBK implements the SBP client protocol. Every SBK instance sends the performance values to SBM using SBP. The SBK initiates the multiple readers and writers to the storage system and then periodically calculates the performance values such as number of bytes and records read and written and the latency values for every record. These performance values are used to calculate the throughput , quartile and percentile of latency values and SLC factors. Each SBK instance prints the performance results such as throughput , quartiles and percentiles of latency values to a local output device and to the Grafana [5] analytics platform via Prometheus [6] monitoring and alerting system to generate the performance graphs. Similarly , the same SBK instance sends the performance values to SBM periodically using the SBP protocol.

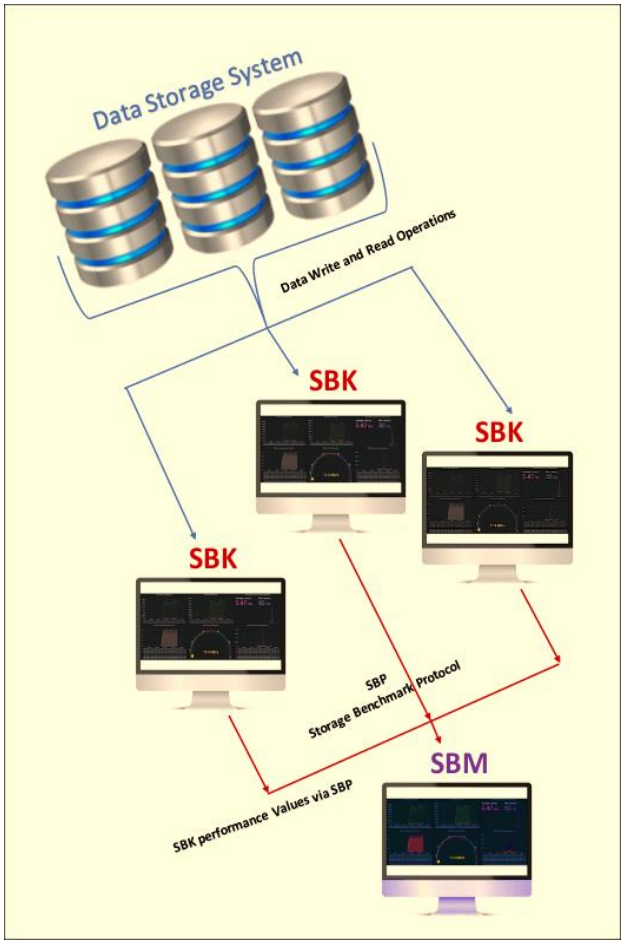

Figure 1: SBK and SBM

Figure 2 shows the message formats of Storage Benchmark Protocol. The SBK (SBP client) initiates the connection to SBM (SBP Server) and then SBM sends the config record to SBK. The config record consists of the storage name for performance benchmarking , Time unit in which latency values are measured. Typically, Time unit will be either a millisecond , microsecond or nanosecond. Minimum and Maximum latency values the SBM can measure and store. The latency values below minimum latency and above the maximum latency are always discarded. After obtaining the config details from the SBM, the SBK can accept it and proceed to send the latencies

records or it can reject the connection due to mismatch of time unit or any other configuration parameters. The latencies record contains the values such as number of readers and writers, the number of records and bytes read and written, minimum and maximum latency values recorded by SBK and the number of discarded latency values too. The latencies record also contains the list/series of the latencies key value pairs in which each pair consists of latency as a key and the count of the number of times the same latency occurred as a value. Transferring a single latencies record from SBK to SBM must be an atomic transaction.

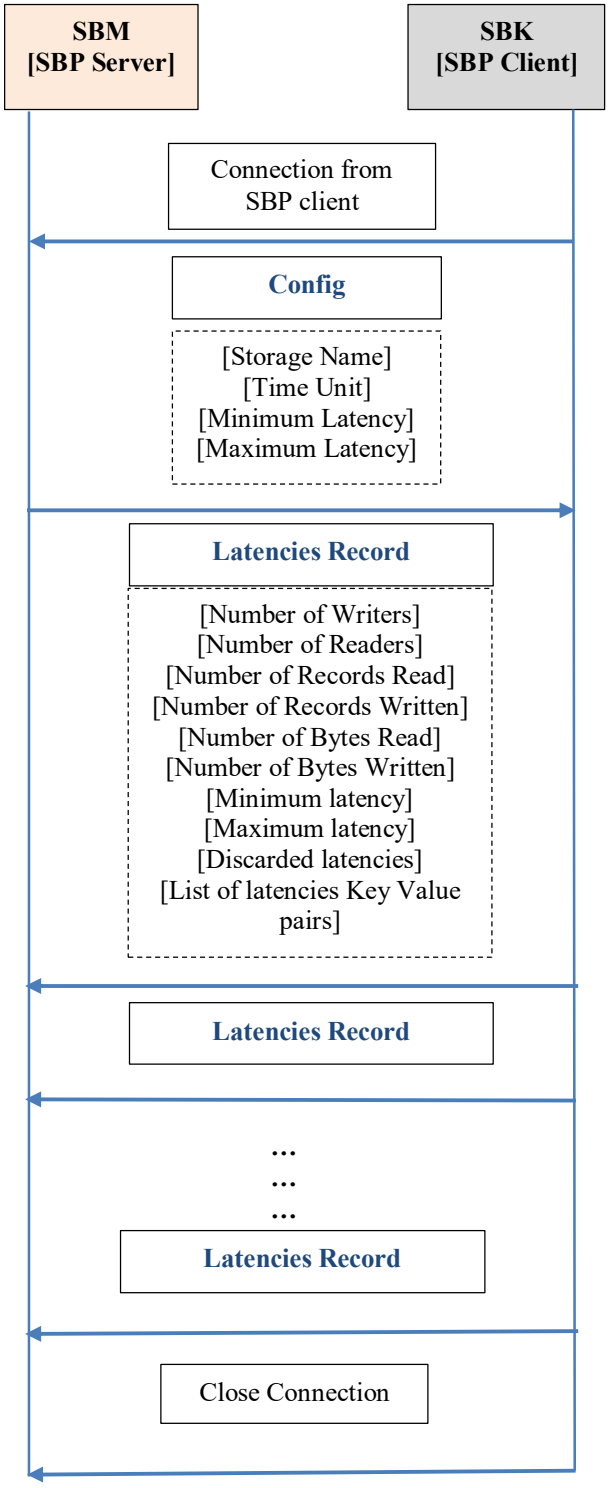

Figure 2: SBP Message formats

The SBM continuously receives the multiple latencies records from single or multiple SBK instances. It consolidate/combines the latencies records and calculates the performance benchmarking values such as throughput in terms of MBs (Mega Bytes)/ Seconds of read and write operations, throughput in terms of Records / Seconds of read and write operations, quartiles and percentiles of latency values and SLC factors. These performance results are periodically delivered to a local output device and to any registered logger and Grafana [5] analytics platform via Prometheus [6] monitoring and alerting systems to generate the consolidated/integrated performance graphs.

Figure 3 shows the design of the SBM. The SBM is a high performance GRPC [7] (Google Remote Procedure Calls) Server. The SBP messages shown in figure 2 are encoded as a Protocol Buffer [8] file. The Protocol buffer compiler (Protoc) [9] generates the template code for the SBP protocol buffer file. The research, depicted in this paper, implements the SBM using the programming language Java [10] version 17. The connection threads shown in figure 3 are the active Java fork join execution threads [11] for the GRPC connections. There will be one active thread per GRPC connection. Each of these threads receives the performance values from SBK via GRPC functions/ methods and enqueues them to a dedicated non-blocking concurrent linked memory queues [12][13]. The SBM has a dedicated thread called "Throughput and Latency Aggregator" to dequeue the performance values from these multiple non-blocking concurrent queues. The SBM uses the APIs(Application Programming Interface) of PerL(Performance Logger) library to calculate the performance results such as throughput and latency percentile values. Both SBK and SBM use common APIs of the PerL library to calculate the performance results. The SBK prints the performance results to a local output device using the Results Aggregation Monitor (RAM) logger and system logger modules shown in figure 3. The performance results are also updated to the metric parameter values of the micrometer subsystem [14] and thus transferred to the Grafana system via Prometheus logger. The Grafana system plots the performance graphs using these performance metrics. A web browser or user applications can fetch these performance metrics too.

The Protocol Buffer Compiler can generate template code for C++, Python and GO Programming languages too. Hence, the existing Java based SBM can be used to fetch the performance values from the SBP clients implemented in C++, Python and GO programming languages.

III. Results And Discussion

In this paper the SBK framework release version 2.0 is used for performance benchmarking. The SBM implemented is part of the SBK framework. The SBM implementation of version 2.0 is inspired/extended from the SBK-RAM (Result Aggregation Manager) of SBK earlier version 1.0. The SBK framework generates the read/write data payloads for benchmarking experiments. The performance benchmarking is conducted using a single file reader thread reading the data from an already existing file in a local file system. The data size is set to 10 bytes for every single read iteration. The hardware and software configurations used for file read performance benchmarking is listed in TABLE1.

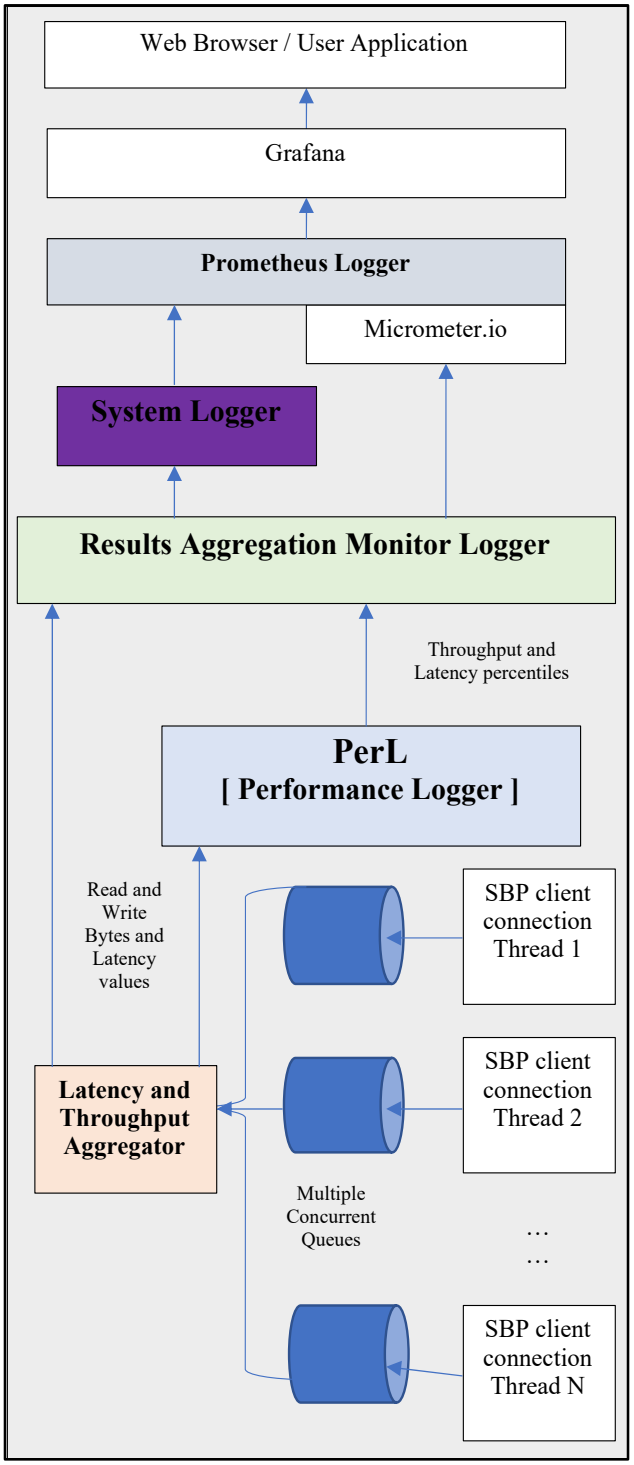

Figure 3: Design of SBM

TABLE I. ARDWARE AND SOFTWARE CONFIGURATION OF THE TEST SETUP

Components	Remarks
System	Mac Book Pro 16-inch , year 2020 model
Number of CPU Cores	8 Intel CPU Cores. Each core is 64 bits.
Main Memory Size	16 GB (Giga Bytes)
Storage Disk	SSD (Solid State Drive) of size 1TB (Tera Bytes)
Operating System	Mac OS Monterey Version 12.5.1
File System	APFS (Apple File System) [15]
SBK and SBM	Version 2.0 [3]
Java Virtual Machine (JVM)	Version 17 [10]
Protocol buffer	Version 3.0 [9]

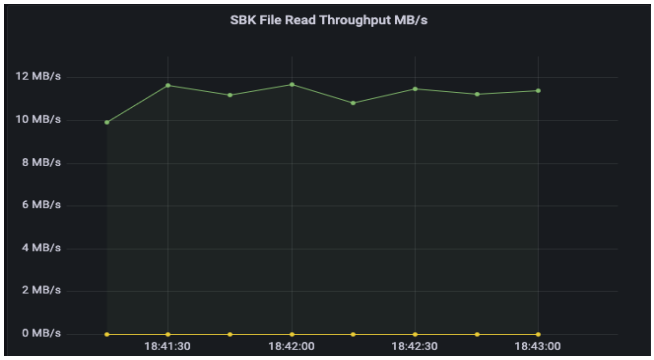

Figure 4: SBK File Read throughput in terms of MB/s

Figure 4 shows the Grafana snapshot of the throughput in terms of MB/second for file read operations by a single instance of SBK. Similarly, Figure 5 shows another Grafana snapshot of the throughput in terms of records/second for the same read operations. The throughput range of 10-12 MB/s and 1 million – 1.2 million records/second are observed for single file reader operations. The latency percentile values of 5th, 10th , 20th , 25th , 30th , 40th and 50th are shown in figure 6. The latency percentile values of 50th, 60th , 70th , 75th , 80th , 90th , 92.5th , 95th, 97.5th , 99th, 99.9th , 99.99th are shown in figure 7. The latency value 700 nanoseconds is observed as 50th percentile and 34.5 microseconds is observed as 99.99th percentile.

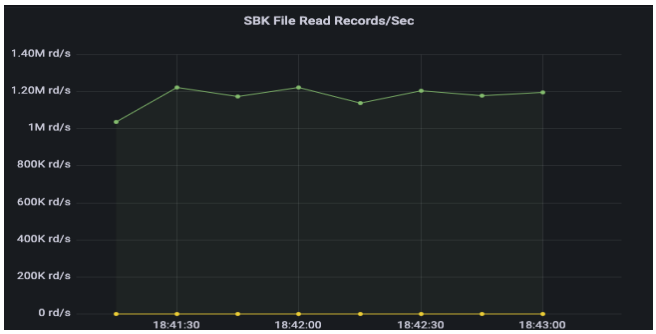

Figure 5: SBK File Read throughput in terms of records/second

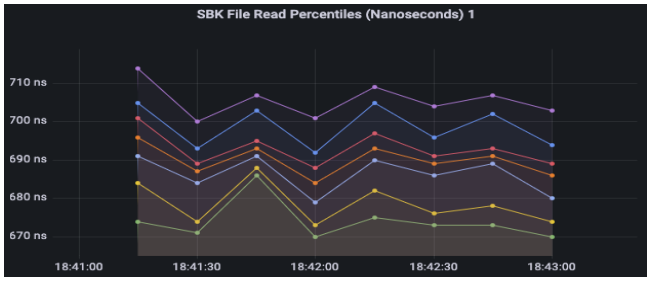

Figure 6: SBK File Read latencies from 5th to 50th percentiles

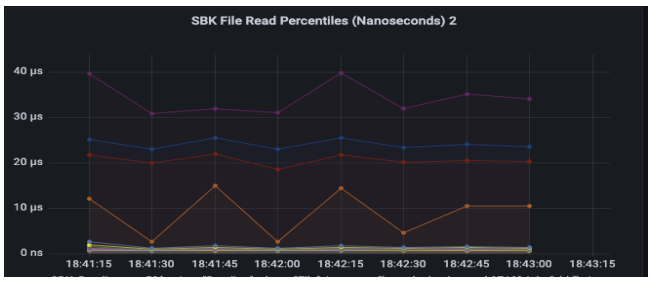

Figure 7: SBK File Read latencies from 50th to 99.99th percentiles

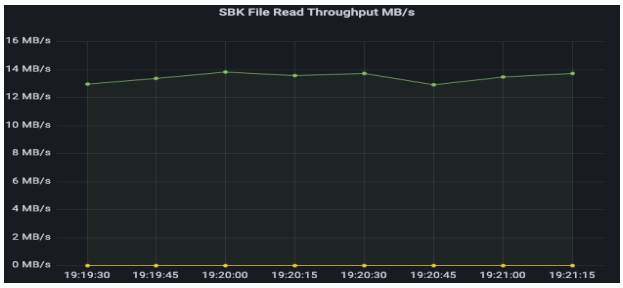

Figure 8: SBM File Read throughput in terms of MB/s

Figure 8 shows the Grafana snapshot of the throughput in terms of MB/second for file read operations by SBM with 2 instances of SBKs with a single reader per each SBK instance. Similarly, Figure 9 shows another Grafana snapshot of the throughput in terms of records/second for the same file read operations. The consolidated throughput range of 12-14 MB/s and 1.2 million – 1.4 million records/second are observed for single file reader operations. The consolidated latency percentile values of 5th, 10th , 20th , 25th , 30th , 40th and 50th are shown in figure 10. The latency percentile values of 50th, 60th , 70th , 75th , 80th , 90th , 92.5th , 95th, 97.5th , 99th, 99.9th , 99.99th are shown in figure 11. The latency value 1.1 microseconds is observed as 50th percentile and 96.6 microseconds is observed as 99.99th percentile. Note that, the combined throughput values are increased and due to an increased number of records the latency percentiles are decreased without any performance degradation.

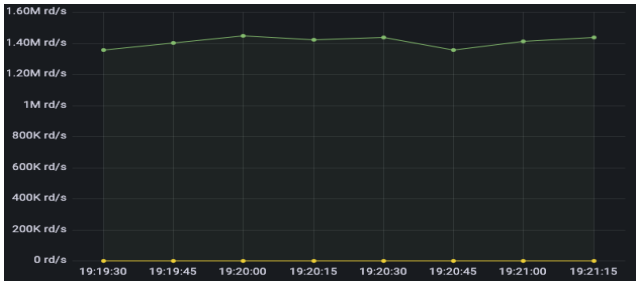

Figure 9: SBM File Read throughput in terms of records/second

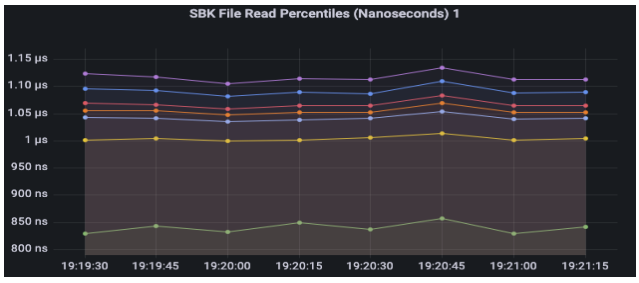

Figure 10: SBM File Read latencies from 5th to 50th percentiles

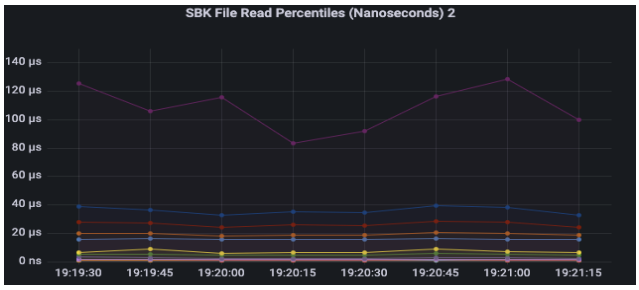

Figure 11: SBM File Read latencies from 50th to 99.99th percentiles

IV. Conclusion

This paper defines and implements the Storage Benchmark Protocol (SBP) and Storage Benchmark Monitor (SBM) as SBP server and SBP client is implemented as part of SBK framework using GRPC framework and Protocol buffers. The implementation of SBM has improved the entire SBK framework deployable as a distributed cluster for any distributed big data storage performance benchmarking. The SBM provides the continuous cumulative performance results by consolidating the performance values from multiple SBK instances. Since the SBP is defined as a Protocol buffer file the Protocol compiler provides a flexibility to implement the SBP client and server (SBM) in C++, Python and GO programming languages along with existing Java implementation. This research work can be extended to implement SBP clients using C++, Python and GO programming languages, thus, SBK framework can support the benchmarking of storage systems with any programming language too. As a future work, both SBM and SBP clients can be extended to deploy remotely on larger distributed systems simultaneously from a single SBM node to multiple SBK nodes.

References

[1] Munegowda K., Sanjay Kumar N.V. "Design and Implementation of Storage Benchmark Kit", In: Emerging Research in Computing, Information, Communication and Applications. Lecture Notes in Electrical Engineering, vol 790, Springer, Singapore. https://doi.org/10.1007/978-981-16-1342-5_5 , 2022.

[2] Storage Benchmark Kit (SBK) : https://github.com/kmgowda/SBK, 2022.

[3] SBK Releases : https://github.com/kmgowda/SBK/releases, October 2022.

[4] Munegowda K., Sanjay Kumar N.V. "SLC: Sliding Latency Coverage Factors for Optimal Performance Benchmarking of Storage Systems", 3rd International Conference for Emerging Technology (INCET), IEEE, https://ieeexplore.ieee.org/document/9825170, May 27-29, 2022.

[5] Grafana Website : https://grafana.com/ , 2022.

[6] Prometheus Website : https://prometheus.io/ , 2022.

[7] GRPC (Google Remote Procedure Calls) framework, website : https://grpc.io/ , 2022.

[8] Protocol Buffers, website : https://developers.google.com/protocol-buffers , 2022.

[9] Protocol buffer compilers, website : https://grpc.io/docs/protoc-installation, 2022.

[10] Java Version 17, Release notes : https://www.oracle.com/news/announcement/oracle-releases-java-17-2021-09-14/

[11] Goetz B (2010) Java Concurrency in practice, Addison-Wesley publications, 9th print.

[12] 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.

[13] Java 7/8 concurrent linked queue , https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html , 2022.

[14] Micrometer IO , website: https://micrometer.io/ , 2022.

[15] Apple File System, website : https://support.apple.com/en-in/guide/disk-utility/dsku19ed921c/mac, 2022.