

# Design of storage benchmark kit framework for supporting the file storage retrieval

Sanjay Kumar Naazre Vittal Rao, Keshava Munegowda

Department of Computer Science and Engineering, Kalpataru Institute of Technology, Tiptur, India

## Article Info

### Article history:

Received Jul 19, 2023

Revised Oct 21, 2023

Accepted Nov 12, 2023

### Keywords:

Benchmarking  
Hadoop distributed file system  
Kafka  
Storage benchmark kit  
XFS file system

## ABSTRACT

An open-source software framework called the storage benchmark kit (SBK) is used to store the system benchmarking performance framework. The SBK is designed to perform any storage client or device using any data type as a payload. SBK simultaneously helps number of readers as well as writes to the storage system of large amounts of data as well as allows end-to-end latency benchmarking for multiple writers and readers. The SBK uses standardized performance measures for comparing and evaluating various storage systems and their combinations. Distributed file systems, distributed database systems, single or local node databases, systems of object storage, platforms of distributed streaming and messaging, and systems of key-value storage are the storage solutions supported by SBK. The SBK supports various storage systems like XFS, Kafka streaming storage systems, and Hadoop distributed file system (HDFS) performance benchmarking. The experimental results show that a proposed method achieves execution time of 65.530 s, 40.826 s and 30.351 s for the 100k, 500k and 1000k files respectively which ensures better improvement than the existing methods such as simple data interface and distributed data protection system.

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



## Corresponding Author:

Sanjay Kumar Naazre Vittal Rao

Department of Computer Science and Engineering, Kalpataru Institute of Technology

Tiptur, India

Email: sanjaynv@gmail.com

## 1. INTRODUCTION

The data storage community plays a crucial part in aiming the large-scale data storage system authenticity as well as mostly used in a large number of applications and systems [1]. The faster development of logistics technologies and e-commerce businesses need automated storage and retrieval systems (AS/RSs). The close storage area, high throughput capacity as well as flexible system structure has widely used in storage systems [2]. Storing and retrieving a large number of data can be retrieved in the cloud anytime and anywhere as well as from any other device, which will be stored in the form of encryption [3]. Due to a large number of nodes, the data can be unfamiliar by storing the data at sometimes [4]. The ancient individual-deep warehouses can store the products or items in an individual-deep framework and utilize a storage as well as retrieval machine provided in the path to serve pair of frameworks. The storage system has a greater storage and retrieval efficiency, however, but wide storage space is employed to store data [5]. Various innovative automated storage and retrieval systems are introduced in past years. The few storages system has faced some drawbacks such as greater basement use, flexible throughput capacity, low-cost investment, double and multiple deep automated storage/retrieval systems, individual vehicle system, and so on [6], [7]. The recent material handling system acquire the compact storage structure named the framework of multiple dimensional storage when compared to the ancient individual deep warehousing system [8]. An open-source

storage benchmark kit (SBK) is an efficient storage performance benchmarking method. The SBK helps performance benchmarking by different mode of implementation named burst mode, rate limiter mode as well as latency mode of end-to-end [9], [10]. This kit helps different storage systems named Apache bookkeeper, Hadoop distributed file systems (HDFS), NATS and streaming, RocketMQ, ActiveMQ Artemis, RabbitMQ, Apache Pulsar, and NSQ as well as streaming storage systems [11], [12]. As well as it supports the database system performance benchmarking named Apache Derby, MySQL, Microsoft SQL as well as SQLite by Java Database Connectivity [13]. In existing Kafka (version 2.3.0), the producer and consumer benchmark tool are utilized for writing and reading performance and helps only producer or writer as well as consumer or reader. Hence, in real-time application, there will be multiple approaches as clear out or reading data from or to an individual Kafka client [14]. The synchronization of various threads minimizes the benchmark tool's robustness and it does not capable to clear out the amount of data to the Kafka client [15]. But in these research, additional process for performance improvement is automatically intricate a process like consumed much memory or computation cost.

The performance benchmarking systems supported both the persistent and distributed key-value stores in the database. SBK can often return the throughput and latencies of read or write results to the Grafana analytics framework by using investigation methods as well as providing performance graphs [16]. Bhat [17] implemented a framework of FUSE using the system of the file, called fumy. This file system was used for the large multimedia efficient storage as small-sized splinters and effectively merge them for retrieval. This system used the service providers for helping quality-of-service (QoS) in downloading when controlling the quality-of-experience (QoE) in streaming. The advantage of this method was not only validating but also evaluating the performance of fumy by combining workloads. However, the fumy may violate the namespace and produce a disparity in flawless fumy working. Liu *et al.* [18] developed a storage connector integration for various storage systems that computes storage systems based on new connector creation. These connector integrations with the service of Globus information transfer permit the movement of information over different storage systems in a mode of "fire-and-forget". This method does not require extended hardware as well as it reduces the minimization of cost. But the extended read operations can cause the performance even if a connector was placed randomly from the storage of the cloud. Rafique *et al.* [19] implemented a flexible and generic data access method of CryptDICE, that runs in a distributed fashion and assures the protection of fine-grained data on the application. This method permits the various types of search and aggregation queries execution by the encrypted data for a large number of various NoSQL databases. The advantage of CryptDICE was a lightweight service, that minimizes the management complexity in the database engine. But this method was not created to generate ciphertexts by which efficient computations can be performed. He *et al.* [20] developed a deep reinforcement algorithm (double and dueling deep Q network) for solving the problem of multi-item retrieval in the system, with basic settings, where, multiple desired items, I/O points as well as escorts are located unevenly. Moreover, a basic compact model of integer programming was developed to evaluate the quality of the solution. The advantage of this method was fully exploited and also integrate existing exact as well as heuristic algorithms, but this method has time complexity in training the process.

Li *et al.* [21] presented a storage method for a new sparse matrix known as the variable blocked- $\sigma$ -SIMD method (VBSF) for the efficient use of vectorization. This method was built by combining adjacent nonzero components under particular standards into different blocks of size as well as padding like zero components. This method provided a storage method feasibility for the basic sparse matrix. The advantage of this method was utilizing the simple vectorization process and easily reusing the data but, the fixed-sized blocks make difficulties in zero padding. Chen and Xu [22] presented an applicable performance comparison of multiple open-source as well as public domain erasure coding libraries namely Jerasure and Intel's ISA-L for RS code and implemented a STAR code. This method aimed to give a guideline data storage practitioner when selecting a suitable erasure code for the functions and methods of storage. This method achieves better performance by using the instructions of SMID as well as providing less computation however STAR code decoding complexity has increased. Munegowda and Kumar [23] implemented and designed the open-source SBK framework. This framework supports the constant and distributed key-value storage performance benchmarking named RocksDB and FoundationDB. The benchmarking results of periodic logging were implemented to the analytic platform of Grafana by the monitoring systems of Prometheus. The advantage of this framework provides stability in both Kafka and HDFS as well as enhanced benchmarking performance. From the overall analysis, the above section contains some limitations such as fumy may violate the namespace and produce a disparity in flawless fumy working, time complexity in training the process, less computation, and fixed-sized blocks that make difficulties in zero padding, the decoding complexity had increased. The proposed SBK with a large number of files is utilized to overcome the limitations of the existing methods. The SBK utilizes the software of a micrometer interface to record results to the monitoring system of Prometheus. The Grafana analytics method acquires the benchmark results from the monitoring system of Prometheus.

The major contribution of this research is listed as follows: initially, storage benchmark kit implementation and design are described in this study. To exhibit the performance benchmark abilities of Kafka, SBK, XFS file system as well as HDFS performance benchmarking are managed. Then, the XFS is the constant file system, which is chosen for benchmarking performance for the file storage retrieval system. The Kafka with 15 divisions, 3 duplicates in that two are coincident duplicates by hardware configuration are used for the benchmarking performance. Also, HDFS is constant and greater achieving in file systems in addition, Kafka is constant and achieves better in groups of distributed storage.

The rest of the paper is arranged as follows: section 2 discussed proposed method. section 3 explains the SBK performance processor. The results and analysis are discussed in section 4 and section 5 presents the overall conclusion of this paper.

## 2. PROPOSED METHOD

This proposed method shows the design and implementation of internal components of the SBK for efficient storage performance. This method includes SBK benchmark, callback (Push) Asynchronous readers, data type handler, storage interface and driver, processor of SBK performance, and finally result logger. Figure 1 shows the design of SBK.

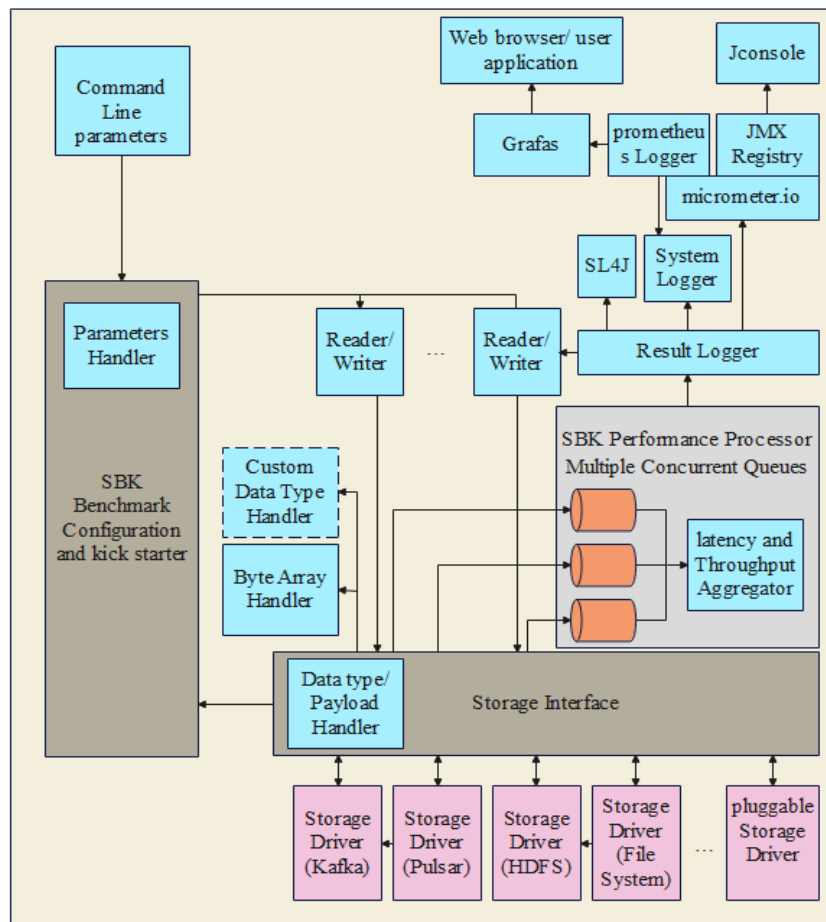


Figure 1. Design of SBK

### 2.1. SBK benchmark

SBK processes and parses user application provided, command line parameters. The SBK designs the multiple readers and writers as well as a constituent called “SBK performance processor”. The SBK initiates callback asynchronous readers for distributed storage system’s messaging platforms named RabbitMQ as well as RocketMQ to file storage system.

## 2.2. Readers/callback (push) Asynchronous readers and writers

The SBK performance processor initiates read-and-write operations of performance benchmarking [24]. These constituents implement the burst mode/max throughput and rate limiter mode to evaluate different latencies under the mode of end-to-end latency as well as records or events rate (throughput). This process can be utilized to examine the amount of time (duration) utilized among the reader and writer as well as identify the similar data record to be utilized.

## 2.3. Data type handler

Data type handler [25] describes data types as well as technique or performance to process data. Byte Array, Protocol buffers, Java NIO Byte Buffer as well as Java String are the illustrations of the data type handlers. A Byte Array is utilized for serialization as well as deserialization of a data. The protocol buffers are unattainable for any situation, in which required to structure as serialize, record-like-typed data in language and platform neutral as well as expandable manner. These majorly utilized for describing communication protocols as well as storage of data.

## 2.4. Storage interface and driver

Storage benchmark kit describes as well as executes the deficiency techniques for the storage interface which is enlarged as well as utilized a usage as well as insertion storage driver to all client or device of the storage. The constituent of the insertion storage driver [26] describes the operations of the read or write of the storage device or client. An individual storage constituent executes a single or variety of detail storage clients or devices. The insertion storage driver either selects accessible or describes current usage data type handlers.

## 3. STORAGE BENCHMARK KIT PROCESSOR

A novel methodology is proposed for the implementation and design of SBK. To exhibit the performance benchmark abilities of Kafka, SBK, XFS file system as well as HDFS are managed by embarrassing the data to the device storage. The XFS is the constant file system, which is chosen for benchmarking performance for the file storage retrieval system. The main idea of the SBK processor is the process of measuring the storage device performance such as how quickly it can read and write the data. The storage benchmark estimates number of operations performed by name node per second than HDFS. Particularly for every operation tested, it relays running time as well as throughput. The storage benchmark kit solves synchronization problems among several readers, and writers as well as reaction threads that are designed consequent to the asynchronous read or write completion operations. These constituents utilize the number of concurrent queues to store a data record enclosing performance value named initial time, finishing time, record numbers as well as byte numbers in the total record numbers to single or multiple operations of read or write. The Java concurrent linked queue gives the thread-safe as well as wait-free/non-blocking application programming interfaces [27] to enqueueing or dequeuing operations. Multiple response read/write completion threads or readers and writers enqueue a execution values to number of concurrent linked queues. But an individual direction called “latency and throughput aggregator” [28] dequeues stored performance values from these number of synchronous linked queues to estimate values of latency as well as throughput. This dequeues values of stored values from number of synchronous linked queues to evaluate values of throughput and latency. The storage benchmark kit handles the response read or write time as the value of latency. The counts of latency are stored in that latency values in the array are utilized for index. The counts that are taken out for latency are stored in the format of the array to be utilized for an index and it is extracted for the latency percentiles evaluation. This evaluation method is utilized in SBK which is simulated by the algorithm of counting-sort with the Big O (maximum latency) time complexity.

### 3.1. Result logger

This constituent receives results like throughput, average and maximum values and percentiles of latency for each agreed time duration from a performance SBK processor constituent. The results are recorded to the device of local output. A SBK utilizes the software of a micrometer interface to record results to the monitoring system of Prometheus. The Grafana analytics method acquires the benchmark results from the monitoring system of Prometheus.

Test distributed file system input-output TestDFSIO tool utilizes a Map-Reduce framework or programming approach and thus achieved much similarity for HDFS tasks. But, SBK utilizes HDFS stream APIs rather than Map-Reduce. The HDFS is fault-tolerant and developed to perform on minimum expensive, martial hardware. HDFS gives maximum throughput data access to application data and is relevant for applications that have large data sets as well as permits streaming retrieve to file system data in Hadoop.

#### 4. RESULTS AND DISCUSSION

In this study, the proposed method is replicated using the SBK with the system requirements. The SBK is evaluated by using Java 8, it is open source in a repository of GitHub as well as this Docker images are available in this hub. SBK release version 0.8 is utilized to file system benchmarking performance as well as HDFS and Kafka is depicted in this method. SBK source code can be improved by details of guide lining the SBK GitHub for an open-source developer as well as a new driver is added for benchmarking the performance of another storage system. The software and hardware configuration of the experimental setup of file system performance benchmarking is operating system: RHEL version 7.4, RAM: 350 GB. The experimental result was evaluated by using the metrics such as accuracy and time.

Accuracy: A measure of ratio of all correct classifications to total number of classifications and it is expressed by (1),

$$Accuracy = \frac{\text{correct prediction}}{\text{Total number of prediction}} \tag{1}$$

Table 1 and Figure 2 represent the number of requested keywords to the mean of time. This concerning the number of keywords comparisons shows predictive performance that consistently improves number of keywords. Table 1 and Figure 2 represents the interaction plot accuracy of the number of candidate keyword to the real keywords. The performance calculation result is shown in Table 1.

Table 1. Candidate keyword number with real keyword of accuracy

No. of keywords	0	1	2	3	4
The mean of time (sec)	1	3	5	7	9

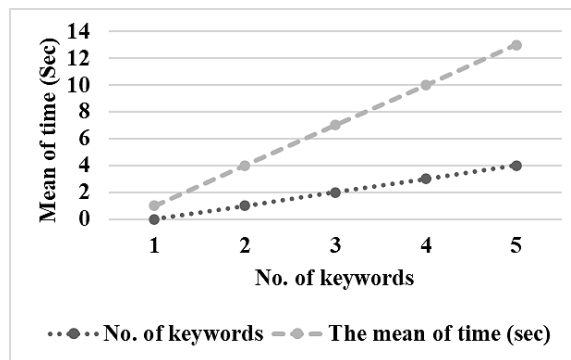


Figure 2. Candidate keyword number with real keyword of the plot for accuracy

Table 2 and Figure 3 represent the number of requested keywords to the mean of time. This regarding the number of keywords in contrast shows forecasting performance consistently improving the keywords. Table 2 and Figure 3 represents the interaction plot accuracy for time matching of the number of candidate keyword to the real keywords. The performance calculation result is depicted in Table 2.

Table 2. Candidate keyword number with real keywords for time matching

No. of keywords	Mean of time (sec)
1	0
3	1
5	2
7	3

Table 3 and Figure 4 represents the number of real keywords vs. significance. This regarding the number of keywords in contrast shows forecasting performance consistently improves the keywords. The performance calculation result is depicted in Table 3.

Table 4 and Figure 5 represents the number of request keyword to the time for the operating matching. This regarding to number of keywords comparisons shows predictive performance that time for matching. The performance calculation result is shown Table 4.

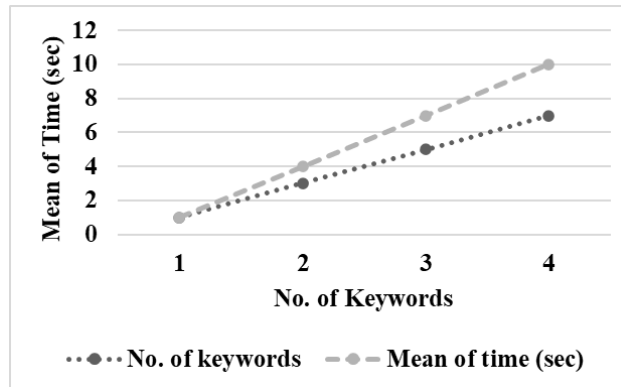


Figure 3. Candidate keyword number with real keywords of plot for time matching

Table 3. Number of real keywords with significance

No. of requested keywords	Time for matching
1	0
3	1
5	2
7	3
9	4
11	5

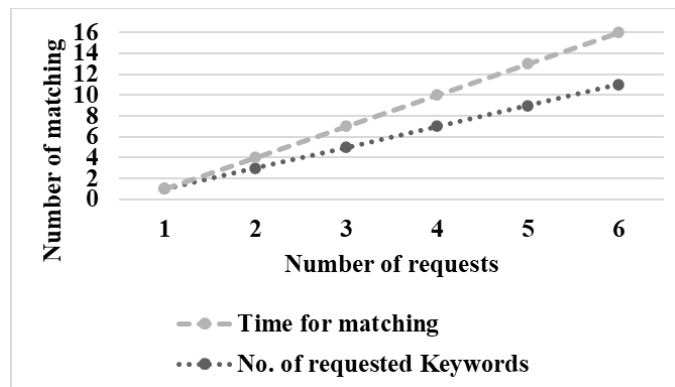


Figure 4. Number of real keywords with significance

Table 4. Number of requests keyword vs. time

No. of requested keywords	Time for matching
1	0
2	1
3	2
4	3
5	4
6	5

In these experimental results, minimizing a value for the configuration parameter enhanced the duration of writing data through embarrassing data to storage device regularly and it minimized the execution process. Therefore, in a testing framework, this method fixes the greater value to 64-bit long and make it as default.

#### 4.1. Comparative analysis

This section shows comparative analysis of proposed storage benchmark kit framework with the recent research methods. This section compared in terms of number of files as well as execution time in seconds is shown in Table 5. The number of files and execution time is more efficient than existing methods. The proposed method's execution time is more efficient than the other methods.

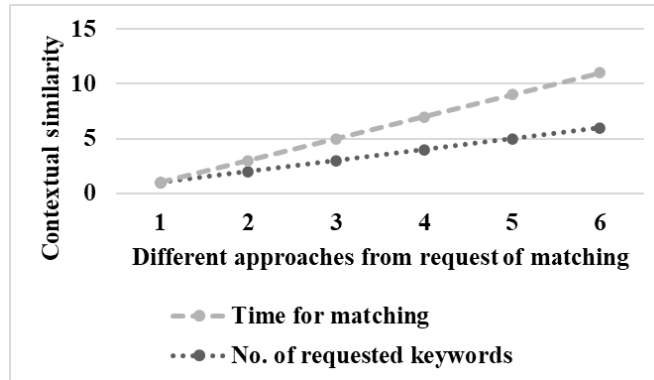


Figure 5. Number of requests keyword vs. time

Table 5. Comparative analysis of the proposed method with the existing method

Method	Number of files	Execution time (sec)
Simple data interface [18]	100k	70.801
Distributed data protection system [19]	500k	60.681
Proposed storage benchmark kit	100k	65.530
	500k	40.826
	1000k	30.351

#### 4.2. Discussion

This section provides the discussion about proposed method and its result comparisons. This section discusses the limitations of the existing methods as well as the proposed method discusses how to overcome these limitations. The simple data interface [18] had the limitation such as extended read operations can cause the performance even if a connector was placed randomly from the storage of the cloud. The flexible CryptDICE [19] does not provide efficient computations and variable blocked- $\sigma$ -SIMD method (VBSF) fixed-sized blocks impact difficulties in zero padding as well as data to reuse are the limitations found from existing methods. In existing Kafka (version 2.3.0), the producer and consumer benchmark tool are utilized for writing and reading performance benchmarking and helps one producer or writer as well as consumer or reader. But, in real-time application, there will be multiple approaches as clear out or reading data from or to an individual Kafka client. The synchronization of various threads minimizes the benchmark tool's robustness and it does not capable to clear out the amount of data to the Kafka client. The proposed SBK framework overcomes these limitations of the existing methods. The SBK is utilized for file storage and retrieval. The proposed method utilized the 1000k number of files to perform the read or write operation with the execution time 30.351 sec where the distributed data protection system [19] have utilized the 500k number of files and it was executed on the 60.681sec to perform the read or write operations respectively.

## 5. CONCLUSION

In this paper, the framework of the SBK design and implementation are presented at a large scale with some readers and writers. This design can be used for solving the synchronization problems among some readers/callback (push) readers, and writers. The SBK design transports standard storage interface APIs, that are increased to contain a storage driver to handle storage client/device performance. The locally mounted file systems, streaming storage platforms, distributed file and messaging systems, key-value, object storage systems, and database systems are the grouped large storage SBK benchmarking. The SBK is utilized as the general framework to manage the benchmarking performance between the same group storage systems. The SBK design is flexible to encompass the non-persistent benchmarking performance in memory

message queues as well as it is utilized to compare the multiple file systems performance. The proposed method achieved the execution time of 65.530 s for 100k number of files, 40.826 s for 500k number of files and execution time of 30.351 s for number of files of 1000k.

The proposed method has some limitations such as multiple encryption operations are required to encrypt each member of an entity individually and it is computationally expensive and intensive. In the future, the proposed storage benchmark kit will extend to check with the real-time applications for enhancing the storage performance.

## REFERENCES





- [1] F. Zammori, M. Neroni, and D. Mezzogori, "Cycle time calculation of shuttle-lift-crane automated storage and retrieval system," *IJSE Transactions*, vol. 54, no. 1, pp. 1–31, Jan. 2021, doi: 10.1080/24725854.2020.1861391.
- [2] X. Xu, X. Zhao, B. Zou, and M. Li, "Optimal dimensions for multi-deep storage systems under class-based storage policies," *Cluster Computing*, vol. 22, no. 3, pp. 861–875, Dec. 2019, doi: 10.1007/s10586-018-2873-9.
- [3] M. Q. Alsudani, H. F. Fakhruddin, H. Abdul-Jaleel Al-Asady, and F. I. Jabbar, "Storage and encryption file authentication for cloud-based data retrieval," *Bulletin of Electrical Engineering and Informatics (BEEI)*, vol. 11, no. 2, pp. 1110–1116, Apr. 2022, doi: 10.11591/eei.v11i2.3344.
- [4] M. M. Arer, P. M. Dhulavvagol, and S. G. Totad, "Efficient big data storage and retrieval in distributed architecture using blockchain and IPFS," in *2022 IEEE 7th International Conference for Convergence in Technology (I2CT)*, 2022, pp. 1–6.
- [5] M. He, Z. Guan, C. Wang, and G. Hou, "Multiple-rack strategies using optimization of location assignment based on MRCSA in miniloader automated storage and retrieval system," *Processes*, vol. 11, no. 3, Mar. 2023, doi: 10.3390/pr11030950.
- [6] S. Geng, L. Wang, D. Li, B. Jiang, and X. Su, "Research on scheduling strategy for automated storage and retrieval system," *CAA Transactions on Intelligence Technology*, vol. 7, no. 3, pp. 522–536, Nov. 2022, doi: 10.1049/cit2.12066.
- [7] Y. Song and H. Mu, "Integrated optimization of input/output point assignment and twin stackers scheduling in multi-input/output points automated storage and retrieval system by ant colony algorithm," *Mathematical Problems in Engineering*, vol. 2022, pp. 1–18, May 2022, doi: 10.1155/2022/5997095.
- [8] A. Edouard, Y. Sallez, V. Fortineau, S. Lamouri, and A. Berger, "Automated storage and retrieval systems: an attractive solution for an urban warehouse's sustainable developmen," *Sustainability*, vol. 14, no. 15, Aug. 2022, doi: 10.3390/su14159518.
- [9] S. Roussanaly *et al.*, "Towards improved cost evaluation of carbon capture and storage from industry," *International Journal of Greenhouse Gas Control*, vol. 106, Mar. 2021, doi: 10.1016/j.ijggc.2021.103263.
- [10] B.-H. Nguyen, T. Vo-Duy, M. C. Ta, and J. P. F. Trovao, "Optimal energy management of hybrid storage systems using an alternative approach of pontryagin's minimum principle," *IEEE Transactions on Transportation Electrification*, vol. 7, no. 4, pp. 2224–2237, Dec. 2021, doi: 10.1109/TTE.2021.3063072.
- [11] Z. Zhu, L. Tan, Y. Li, and C. Ji, "PHDFS: optimizing I/O performance of HDFS in deep learning cloud computing platform," *Journal of Systems Architecture*, vol. 109, Oct. 2020, doi: 10.1016/j.sysarc.2020.101810.
- [12] H. Nakanishi *et al.*, "Design for the distributed data locator service for multi-site data repositories," *Fusion Engineering and Design*, vol. 165, Apr. 2021, doi: 10.1016/j.fusengdes.2020.112197.
- [13] A. Basuki and A. Adriansyah, "Response time optimization for vulnerability management system by combining the benchmarking and scenario planning models," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, no. 1, pp. 561–570, Feb. 2023, doi: 10.11591/ijece.v13i1.pp561-570.
- [14] H. Asfa and T. Javdani Gandomani, "Software quality model based on development team characteristics," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, no. 1, pp. 859–871, Feb. 2023, doi: 10.11591/ijece.v13i1.pp859-871.
- [15] S. Myint and W. Wichakool, "A simple faulted phase-based fault distance estimation algorithm for a loop distribution system," *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*, vol. 25, no. 1, pp. 14–24, Jan. 2022, doi: 10.11591/ijeecs.v25.i1.pp14-24.
- [16] S. Baek, "System integration for predictive process adjustment and cloud computing-based real-time condition monitoring of vibration sensor signals in automated storage and retrieval systems," *International Journal of Advanced Manufacturing Technology*, vol. 113, no. 3–4, pp. 955–966, Jan. 2021, doi: 10.1007/s00170-021-06652-z.
- [17] W. A. Bhat, "FUSE based file system for efficient storage and retrieval of fragmented multimedia files," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 8380–8389, Nov. 2022, doi: 10.1016/j.jksuci.2022.08.018.
- [18] Z. Liu, R. Kettimuthu, J. Chung, R. Ananthakrishnan, M. Link, and I. Foster, "Design and evaluation of a simple data interface for efficient data transfer across diverse storage," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 6, no. 1, pp. 1–25, Mar. 2021, doi: 10.1145/3452007.
- [19] A. Rafique, D. Van Landuyt, E. Heydari Beni, B. Lagaisse, and W. Joosen, "CryptDICE: distributed data protection system for secure cloud data storage and computation," *Information Systems*, vol. 96, Feb. 2021, doi: 10.1016/j.is.2020.101671.
- [20] J. He, X. Liu, Q. Duan, W. K. (Victor) Chan, and M. Qi, "Reinforcement learning for multi-item retrieval in the puzzle-based storage system," *European Journal of Operational Research*, vol. 305, no. 2, pp. 820–837, Mar. 2023, doi: 10.1016/j.ejor.2022.03.042.
- [21] Y. Li *et al.*, "VBSF: a new storage format for SIMD sparse matrix–vector multiplication on modern processors," *The Journal of Supercomputing*, vol. 76, no. 3, pp. 2063–2081, Mar. 2020, doi: 10.1007/s11227-019-02835-4.
- [22] R. Chen and L. Xu, "Practical performance evaluation of space optimal erasure codes for high-speed data storage systems," *SN Computer Science*, vol. 1, no. 1, Jan. 2020, doi: 10.1007/s42979-019-0057-1.
- [23] K. Munegowda and N. V. Sanjay Kumar, "Design and implementation of storage benchmark kit," in *Lecture Notes in Electrical Engineering*, vol. 790, Springer Singapore, 2022, pp. 45–62.
- [24] P. Salzmann, F. Knorr, P. Thoman, P. Gschwandtner, B. Cosenza, and T. Fahringer, "An asynchronous dataflow-driven execution model for distributed accelerator computing," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2023, pp. 82–93, doi: 10.1109/CCGrid57682.2023.00018.
- [25] M. Hunko, V. Tkachov, O. Liashenko, and J. Rabcan, "Application architecture for obtaining data from scientometric database," in *2022 IEEE 3rd KhPI Week on Advanced Technology (KhPIWeek)*, Oct. 2022, pp. 1–4, doi: 10.1109/KhPIWeek57572.2022.9916398.
- [26] C. Chang *et al.*, "MetaScenario: a framework for driving scenario data description, storage and indexing," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 2, pp. 1156–1175, Feb. 2023, doi: 10.1109/TIV.2022.3215503.







- [27] J. D. Hughes, M. J. Russcher, C. D. Langevin, E. D. Morway, and R. R. McDonald, "The MODFLOW application programming interface for simulation control and software interoperability," *Environmental Modelling & Software*, vol. 148, Feb. 2022, doi: 10.1016/j.envsoft.2021.105257.
- [28] Y. Zhu and M. Xu, "Enhancing network throughput via the equal interval frame aggregation scheme for IEEE 802.11ax WLANs," *Chinese Journal of Electronics*, vol. 32, no. 4, pp. 747–759, Jul. 2023, doi: 10.23919/cje.2022.00.282.

## BIOGRAPHIES OF AUTHORS



**Sanjay Kumar Naazre Vittal Rao**     is Associate Professor at College of Computer Science and Engineering, Visvesvaraya Technological University, Karnataka, India. He Holds a M.Tech. degree in Computer Science and Engineering. Currently he is research scholar under VTU, India. His research areas are big data, storage system benchmarking. His research interests include storage systems, performance analysis, analytics, distributed systems. He can be contacted at email: sanjaynv@gmail.com.



**Keshava Munegowda**     received the Ph.D. degree in computer science from the University of Visvesvaraya Technological University, Karnataka, India. Currently he is working as Vice President, SecDB Engineering, Goldman Sachs, Bengaluru, Karnataka, India. He served as system architect/technical manager and an individual contributor too. 18+ years of experience in design, development of file systems, storage systems, cluster infrastructure development, software defined storage/streaming storage, big data systems and network authentication and security protocols. He has authored or coauthored more than 20 publications: 8 US patents. His research interests include storage systems, performance analysis, analytics, distributed systems, algorithms. He can be contacted at email: kmgowda@gmail.com.