ISSN 1816-6075 (Print), 1818-0523 (Online)
Journal of System and Management Sciences
Vol. 13 (2023) No. 1, pp. 605-619
DOI:10.33168/JSMS.2023.0130

Dynamic Load Balancing in Distributed Storage Systems using Modified Whale Optimization Techniques

K.J. Rajashekar [1], Channakrishnaraju [2]

[1] Sri Siddhartha Academy of Higher Education, Tumakuru, Department of Information Science and Engineering, Kalpataru Institute of Technology, Tiptur. 572201, India

[3] Department of Computer Science and Engineering, Sri Siddhartha Institute of Technology, Tumakuru. 572105, India

rajkit2006@gmail.com (Corresponding Author); rajuck@ssit.edu.in

Abstract. A large amount of data and the synchronization issue between various consumers or producers are handled by using distributed storage systems, which is a complex task. As a result, this research proposal presents payload benchmarking with dynamic load balancing in distributed streaming storage systems using modified Whale optimization techniques. The main objective of the optimization technique is to analyze the benchmarking data and find the failure prediction of Kafka benchmarking. The configuration files of the benchmark are taken as input to build the Kafka setup to capture the read and write latency. Kafka is a model where the number of producers is limited but the number of consumers is growing at an exponential rate. Kafka is a distributed system comprised of servers and clients which is used for streaming processors in real-time messaging systems. The modified Whale optimization algorithm is implemented to find the solution to an optimization problem, especially with incomplete or inaccurate data. The performance of the Kafka and DevOps systems is validated by the parameters of network latency as 400ms, Execution time as 980ms, Memory size as 390kb, and Bandwidth as 380kb. The performance of the benchmarking tool for distributing the streaming storage system has been improved to achieve the maximum possible throughput from the streaming storage system.

Keywords: Load Balancing, Kafka, Modified Whale Optimization Algorithm, Benchmarking, Distributed Storage Systems

1. Introduction

The data balancing between various virtual machines within an optimum time is efficient for the performance of load balancing in distributed storage systems. The large amount of information is processed efficiently by using Kafka which is an open-source distributed platform for streaming data. The various servers are using Kafka and perform fast due to data stream decoupling and resulting in less latency. The issues of distributed systems like replication, node failures, and ensuring data integrity are performed efficiently by using Kafka. The main objective of using Kafka is for dynamic payload benchmarking for distributed storage systems with a modified Whale load balancing optimization algorithm. The streaming data from various distributed storage systems is processed based on a processor like Kafka (Xingjun et al., 2020). The performance of Kafka is validated by the parameters of latency, bandwidth, memory size, and execution time to improve the performance of business requirements (Sfaxi and Aissa 2021). The main objective of proposing the Kafka broker is too dynamic payload benchmarking for distributed storage systems (Barba-Gonzalez et al., 2020). The users or producers send various requests to the consumer hence to maintain the messages between the producer and the consumer is managed by using the broker called Kafka where a large amount of data is portioned into various systems to make the process efficient (Brandon et al., 2020). The Kafka monitors the traffic between the user and the server requests (Shafiq et al., 2021) and a large amount of information between the server and user is managed by using optimization algorithms (Talaat et al. 2020). Benchmarking is the process of comparing any organization to the best organization to make it possible to improve the output at more than the best present with high performance, more quality, and low cost. The large amount of data produced from the Internet of Things is required sufficient computation power, and storage to balance the data and reduce the bandwidth, and complexity of the data is giving by using optimization algorithms (Peng et al., 2020). The various dimension of workload is considered for benchmarking of the data by using the open source framework namely Kafka (Henning and Hasselbring 2021). The Kafka gives efficient results compared to the existing Spark, Flink, Storm, Apex, and Beam (Devaraj et al., 2020) due to the qualities of high scalability, durability, and speed. The multi-objective load balancing techniques like Grasshopper Optimization Algorithm (GOA), Particle Swarm Optimization algorithm (PSO), and Grey Wolf Optimization algorithm (GWO) algorithms, are presented in the existing methods but due to local optimum issues, the performance is reduced (Neelima and Reddy 2020). The load balancing is a complex task for a large amount of data to improve the performance of the model (Balaji et al., 2021). The minimization of the workload and resulting relevant data reduce the latency problems and improve the performance (Khriji et al., 2022). The performance metrics like load balancing, power consumption, and resource utilization are resulted efficiently by using the modified Whale optimization algorithms (Barba-Gonzalez et al., 2020; Goyal et al., 2021). The growth of the data in servers and networks is managed by using the maximum amount of resources, improving scalability, eliminating disruptions, and reducing the over-supply (Chakraborty et al., 2021). The main contributions of the proposed Kafka method using MWOA is represented below:

1. The benchmark configuration files are taken as the input to analyze the dynamic payload benchmarking for improving the performance of the systems.
2. The huge amount of data is preprocessed by using Kafka which is an open-source stream processing platform. Kafka gives efficient results for fault tolerance and scalability in distributed storage systems and enables the streaming of data.
3. The preprocessing data is balanced by using the modified Whale optimization algorithm which is an efficient technique for load balancing mechanism evaluated to reduce the drawbacks present in the existing optimization algorithms like PSO, GOA, and GWO.
4. The performance is validated in terms of latency, bandwidth, memory size, and read size of Kafka and resulting the improved results.

The following research paper is organized as a review of the existing works in Section 2, and the proposed methodology is explained in Section 3. Section 4 evaluates the results of the proposed algorithm, Section 5 illustrated the comparative analysis, and the conclusion of the paper is given in section 6.

2. Literature Review

U.K. Jena, et al (2020) developed a novel dynamic load balancing using the hybridization Modified Particle Swarm Optimization (MPSO) algorithm. The performance of the machine was improved by balancing the load between various virtual machines by reducing the waiting time for tasks. The static load balancing methods gave efficient results with low fluctuation of load in the virtual machine. The static load balancing does not give efficient results by varying the loads unpredictably during run time and sufficient memory was required for the server to create the virtual machines. Veeramanikandan, et al (2020) presented a novel Distributed Deep Neural Network and Data Flow for big data to result in reduced network, latency, and service. The Deep neural network reduces the workload of cloud and network congestion by taking decisions at different levels in a distributed manner. The main drawback of the Deep Learning algorithm was consumed more computational time to process the data. The Data Flow of Distributed network performance was improved by edge computing methodology. Maycon V. Bordin, et al (2020) developed efficient Data Stream Processing Systems to improve the computation performance of distributed storage systems. The performance of the benchmark distributed storage systems was improved by using the apache storm and spark streaming analysis. The workflow was characterized based on the processing cost, input size, occupation of memory, and selectivity and improved the performance of computation but in the case of the high values, the computational time and cost were improved.

Giselle Van Dongen, et al (2020) presented the frameworks like Flink, Spark Streaming, and Structured streaming to improve the benchmarking scalability. The scaling direction and cluster layout were used as influencing factors to improve the scaling efficiency. The scalability of the framework, bottleneck throughput, and design framework were influenced by using the preprocessing characteristics. The performance of the frameworks did not result in suitable results in the scalability of benchmarking. Chunlin, et al (2022) presented a geographically distributed storage system to reduce the transmission time and also reduce the cost of bandwidth. The load balancing was done optimally by using the geographically distributed cloud. The main objective of using geographical systems is to have high performance with less time and optimal cost. The Floyd algorithm was used to minimize the cost of data transmission bandwidth. The limitation of using geographical systems is not suitable for more than the limited data and results in less performance. G.Annie Poornima Princess, et al (2021) presented Harries Hawks Optimization Algorithm (HHOA) and Pigeon-inspired optimization algorithm for load balancing by utilizing optimal resources with fewer periods. The JAVA Net beans were used to implement the frameworks to access the number of tasks and the performance. The proposed optimization algorithms minimize the computational time and result in optimal solutions to achieve efficient load balance. The incoming requests are balanced by using the Hawks algorithm by reducing the overload but the performance of the model needs to be improved with optimum cost.

Arfa Muteeh, et al (2021) presented a Multi-resource load balancing algorithm (MrLBA) for reducing the load balance of the model and used the Ant Colony Optimization algorithm (ACO). The proposed ACO optimization algorithm results in efficient performance in load balancing of benchmark data with less cost. The data was preprocessed to minimize the bottleneck tasks and better resource allocation was done. The main drawback of the ACO was used more resources to maintain the balancing of the data to result in optimal solutions.

3. Methodology

The payload benchmarking with optimization load balancing algorithm on Kafka processor is proposed to result in efficient load balancing for benchmark dataset. Kafka gives efficient results for fault tolerance and scalability in distributed storage systems and enables the streaming of data in real time. Modified Whale optimization algorithm gives optimal solution to a complex and difficult problem

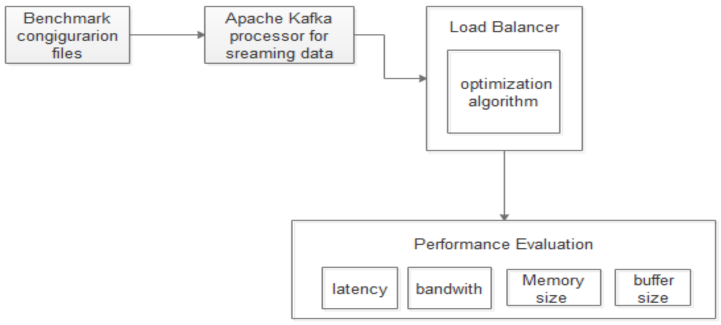

Fig. 1: Flow of Dynamic Payload Benchmarking

3.1. Dataset

The benchmark configuration files are used as input for the load balancing in distributed streaming storage systems for training and testing. The benchmark datasets are most significant for high-performance load balancing. The performance results of the consumer and producer are benchmarked and validate the message size, and batch size for Kafka configuration.

3.2. Data Preprocessing

A large number of messages in real-time is collected and pre-processed by using Kafka which is an open-source stream processing platform. Kafka gives efficient results for fault tolerance and scalability in distributed storage systems and enables the streaming of data in real time. The records are published by the producers and consumers subscribe to the topics in that records where every record consists of a Key-Value pair. The intermediary between the producer and consumer to build effective distribution is an important task in Kafka. In case of a high amount of data, the data is portioned into various partitions, for every partition the read and write operations occur parallel. The input information from producers is stored in the format of log files with the topic name and partition number up to 1Gb, after that, the new segment with log, time, and index is created. The timestamp segment is used for deleting the data automatically when the limit of the time stamp is expired by using the Time To Live (TTL) operation. Based on the replication factor if the replication factor is more than 1 for a topic the partition of data takes place. The partition of data is two types: follower and leader like a master-slave system and only one leader is present in every partition in a period, the rest of the partitions are followers. The write and read operations are performed only through the leader node. Every cluster of Kafka consists of a controller to assign the partitions for the followers and leader. In this case, all the operations of read and write are performed by the leader and the followers are in an idle position hence a synchronization of data takes place to send the input data to the leader and the followers parallel by using the sync replicas for partitions (ISR) where the value of ISR is calculated by replication factor minus 1. The main concept of Kafka is producers and consumers where the producer application produces the data to provide the data for some other applications and the consumer is a feature of Kafka that allows multiple consumers to read similar messages.

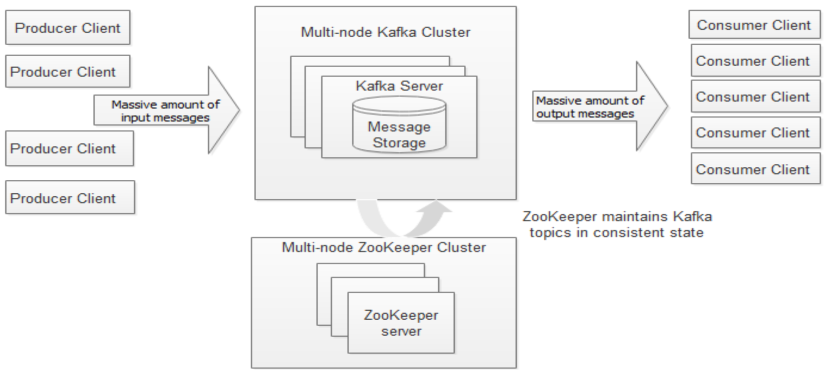

Figure 2: Architecture of Apache Kafka

Kafka receives various messages from the producer and sends them to the consumer where every message is having the attributes of the producer and consumer. The producer is sending the message to Kafka with some acknowledgment. In some cases of acknowledgments, the producer sends the message and does not wait for delivery of the message is zero acks, the producer sends the message and waits for successful delivery from the leader is ack-1 and ack=-1 where the message is sent by the producer and waiting for confirmation from leader and ISR. The consumer receives the group of messages from the cluster of Kafka where every consumer group consists of a specific group id, stored in the offset with the name of the topic. The offset of every consumer group is stored separately then the new consumer gets messages from the offset stored in the file of the system.

The time taken for a message to move from the producer to the consumer is calculated as the latency of Kafka in the Kafka cluster. The start time of a message from the producer until the message is appended to log partitions is treated as the production time. The Kafka replicates the message for fault tolerance then the message is ready to consume, the committed phase is called commit time. The commit time is calculated up to the consumer receiving the message from the cluster. The maximum data size of the accumulated message in one batch represents the batch size and the maximum number of batches with limited space as spatial batch size is denoted as B_s and the maximum time taken to construct the batch is the temporal batch size(B_T). The configuration of spatial size and temporal size makes the Kafka producer send the batch. The value of B_T is constant and the value of B_s

is varying then the mean of the latency L_K is changed. The DevOps model is implemented for developing the new products quickly and maintain the existing information. The comprehensive view that results the business value of DevOps by using the Key Performance Indicators (KPI) and results efficient latency, memory size and bandwidth efficiently.

3.3. Load Balancer

The huge amount of data produced in the real world is balanced by using optimization load-balancing algorithms like the modified Whale algorithm. The input tasks are uniformly split among the leader and the followers to make the process simple and fast. The main objective of load balancing is to reduce the overloaded tasks from one machine and assigned them to another system to improve the efficiency, and throughput, and reduce the time of the machine. The Whale optimization algorithm results in efficient results in balancing the data across various machines. The Whale algorithm is designed based on the algorithms like a swarm, bird flocking, and using mathematical models. The performance metrics like load balancing, power consumption, and resource utilization are resulted efficiently by using the modified Whale optimization algorithms. The input benchmark configuration files are allocated to various systems to store and balance the data from producer to consumer the Whale optimization algorithm is proposed. Whale optimization is a meta-heuristic technique proposed based on the hunting behavior of humpback whales by using the mechanism of a bubble net to chase their prey. The updated Whale Optimization Algorithm (WOA) is Modified Whale Optimization Algorithm (MWOA) which is incorporated with the levy flight using Mantegna's algorithm to give optimal solutions. The MWOA is generating random configuration files which produce radical networks to reduce the overloading of the data.

3.3.1. Encircling Prey

The WOA works on finding the optimal solution for balancing the data by encircling all other prey and updating the optimal solution. The mathematical expression of encircling behavior is given in equations-(1) and (2).

$$\vec{D} = |\vec{c}.\vec{X}^*(t) - \vec{X}(t)| \tag{1}$$

$$\vec{X}(t+1) = \vec{X}^*(t) - \vec{A}.\vec{D} \tag{2}$$

Where the number of current iterations is represented as t, X is the present solution, X^* is the best solution in the overall information, and c, A is the coefficient vectors calculated by the equations (3) and (4) respectively.

$$\vec{A} = \overrightarrow{2a}.\vec{r} - \vec{a} \tag{3}$$

$$\vec{C} = \overrightarrow{2r} \tag{4}$$

3.3.2. Bubble-Net Method

The shrinking method of encircling and spiral position updating is used for performing bubble-net attacks. The Shrinking encircling is calculated by the equations of (5) and (6) where $\vec{a}$ is linearly minimized as 2 to 0 and the value of $\vec{A}$ is limited between [-1,1]. The current solution is updating randomly with the best solution. The current solution updated with a new solution is giving a new position by using the Spiral Updating Position.

$$\vec{D} = |\vec{X}^*(t) - \vec{X}(t) \tag{5}$$

$$\vec{X}(t+1) = \vec{D}|.e^{bl}.\cos(\pi l) + X^*(t) \tag{6}$$

Where b defines the logarithm spiral constant, l is a number generated randomly in between the range [-1,1]. The parameter p selection is given to balance the two models where the value of p is 0.5 for WOA which is represented in the equation-(7).

$$\vec{X}(t+1) = \begin{Bmatrix} \vec{X}^*(t) - \vec{A}.\vec{D}, \quad if\ p < 0.5 \\ \vec{D}|.e^{bl}.\cos(2\pi l) + X^*(t), \quad if\ \ p \geq 0.5 \end{Bmatrix} \tag{7}$$

Where p is a random value ranging between [0,1].

3.3.3. Search for Prey

The whales move randomly to each other and find new prey where the random search operation is explored when $|A| \geq 1$. In the present information, the present solution is updated with the random solution in place of the best solution which is measured by the equation- (8) and (9).

$$\vec{D} = |\vec{c}.\vec{X}_{rand}(t) - \vec{X}(t)| \tag{8}$$

$$\vec{X}(t+1) = \vec{X}_{rand}(t) - \vec{A}.\vec{D} \tag{9}$$

Where $\vec{X}_{rand}$ is the random solution for present information.

The WOA consists of some problems like premature phenomenon convergence leading to local optimization and the global search ability is defined which reduces the performance of WOA. The performance is improving by proposing the MWOA by improving the exploration ability. The exploration ability of further iteration is improved by defining a new parameter B to control the data where B is calculated by the equation-(10).

$$B = a + 2.r - 2 \tag{10}$$

Where a is linearly minimized from 2 to 0, and r is a random value ranging between [0,1].

The encircling prey is modified in MWOA where the positions of search agents are improved to the best searching agent. The convergence of premature makes it easy to find the local solution and enables the agents to reduce the stagnation where the modified encircling is measured by the equation-(11).

$$X_i(t+1) = X_{r_1} + 0.5(X_{r_2} - X_{r_3}) \tag{11}$$

Where $r_1, r_2, \text{ and } r_3$ are randomly selected agents, where $r_1 \neq r_2 \neq r_3 \neq i$.

4. Results and Discussions

The payload benchmarking with dynamic load balancing in distributed storage systems on Kafka is efficient and a large amount of data is balanced by using the Modified Whale optimization algorithm. The performance of the model is validated on the virtual machine consisting of 10Gb Random Access Memory, 36 Gb virtual storage, 2.4 processor, and 227Gb HD storage. The comparative analysis is performed between various load optimization algorithms like Grasshopper Optimization Algorithm (GOA), Particle Swarm Optimization algorithm (PSO), Grey Wolf Optimization algorithm (GWO), and Modified Whale Optimization Algorithm(MWOA) with different models like Spark, Flink, Storm, and Kafka where the proposed Kafka model with MWOA results in efficient results compared to the existing methods. The performance is validated by the parameters of latency, memory size, execution time, and bandwidth.

The performance of latency is measured by the time certain changes are caused in load balancing of the data where the latency is minimized then the efficiency of the system is improved. The memory space utilized by the model to perform a specific task is measured as the size of the memory, the usage of fewer memory results in more efficiency and less time. The time taken to perform an input task is considered the execution time of the model. The bandwidth is the maximum amount of data transferred across the given path that has resulted as the bandwidth of the model. The performance of latency, Memory size, Execution time, and Bandwidth are calculated by the equations- (12), (13), and (14).

$$Latency = \left(\frac{\beta_i}{k_j}\right)(u(t)+1) \tag{12}$$

Where $u(t)$ is the current number of workloads in time t, k_j denotes the number of instances per unit, β_i is the instance from the source code.

$$Memory\ size = \frac{1}{PM+VM}\left[\sum_{I=1}^{PM}\sum_{J=1}^{VM}\frac{1}{2}\left(\frac{CPU\ Utilized}{CPU} + \frac{Memory\ Utilized}{Memory}\right)\right. \tag{13}$$

Where VM is the virtual machine with i instances

$$Execution\ time = \frac{task}{processimg\ speed\ of\ VM},$$
$$Processing\ speed = \frac{speed\ of\ task}{number\ of\ VM} \tag{14}$$

4.1. Quantitative Evaluation

This section evaluates the Kafka model by using MWOA which gives effective results in dynamic payload benchmarking with input benchmark configuration files. The comparative analysis is performed between various load optimization algorithms like GOA, PSO, GWO, and MWOA with different models like Spark, Flink, Storm, and Kafka where the proposed Kafka model with MWOA results in efficient results compared to the existing methods. The performance of the Kafka model with the MWOA optimization algorithm is validated by the parameters Latency, Execution time, Memory size, and Bandwidth. The load balancing of the information between the producer and consumer in the Kafka cluster is balanced efficiently by using the MWOA optimization algorithm. The improved performance results in the latency, Execution time, Memory size, and Bandwidth.

The performance of the latency, Execution time, Memory size, and Bandwidth by using various processing models and optimization algorithms have resulted in the table-1. The performance is measured by the transaction that takes place in the Kafka cluster between the consumer and the producer. The results evaluated the latency as 400ms, Execution time as 980ms, Memory size as 390kb, and Bandwidth as 380kb.

Table 1: Experimental results of various performance parameters

		Latency(ms)		
Models	GOA	PSO	GWO	MWOA
Spark	4900	5000	5100	4800
Flink	5300	5400	4900	4700
Strom	5100	4900	5100	4600
Kafka	4500	5100	5300	4000
		Memory Size(kb)		
Models	GOA	PSO	GWO	MWOA
Spark	570	520	720	530
Flink	540	630	620	560
Strom	490	658	550	470
Kafka	450	480	440	390
		Execution time(ms)		
Models	GOA	PSO	GWO	MWOA
Spark	1490	1500	1510	1480
Flink	1530	1540	1490	1470
Strom	1510	1490	1510	1460
Kafka	1450	1510	1530	980
		Bandwidth(kb)		
Models	GOA	PSO	GWO	MWOA
Spark	530	520	630	480
Flink	560	545	540	530

Strom	490	490	530	470
Kafka	450	420	410	380

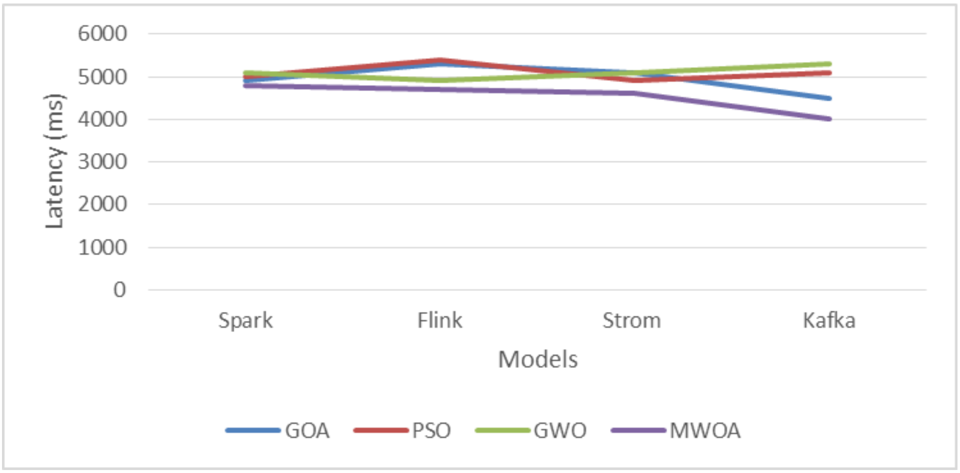

Fig. 3: Experimental Results of Latency Performance

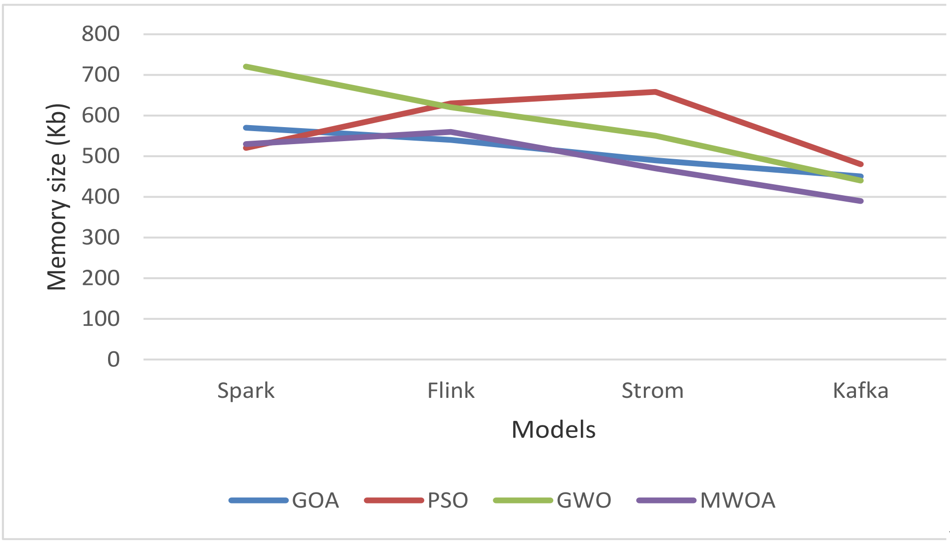

Fig. 4: Experimental Results of Memory Size

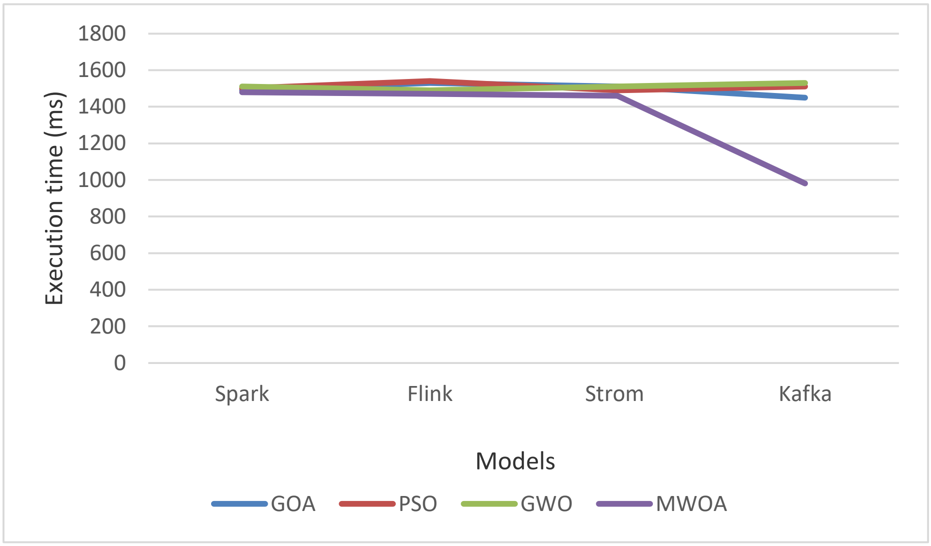

Fig. 5: Experimental results of the execution time

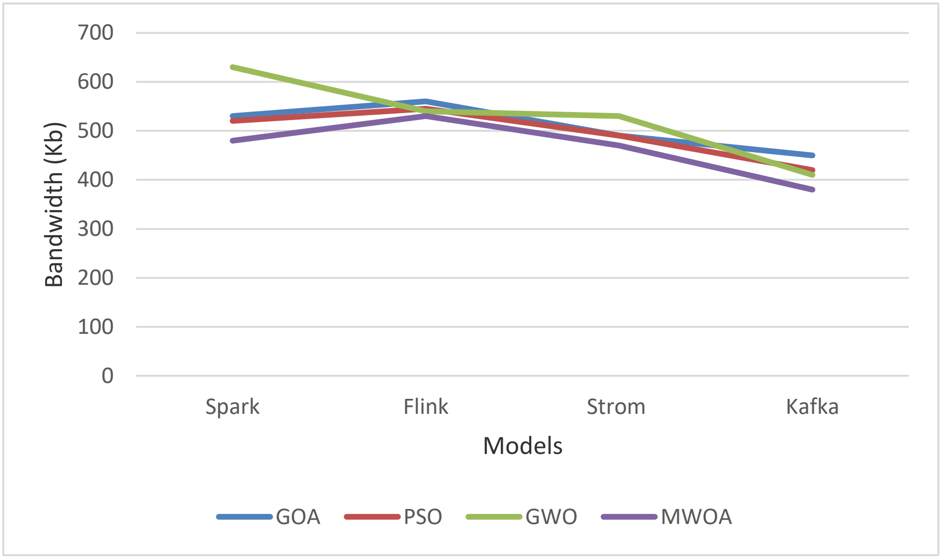

Fig. 6: Experimental Results of Bandwidth Performance

The payload benchmarking on Kafka using the MWOA optimization algorithm for distributed storage systems results in efficient performance which is validated by the parameters Latency, Memory size, Execution time, and Bandwidth. The performance of Kafka using MWOA is compared by using various existing methods resulting in the table-1 and the graphical representation of the results is presented in figures- (3), (4), (5), and (6).

5. Comparative Analysis

The Comparative Analysis takes place between the existing algorithms like MPSO, HHOA, and ACO optimization algorithms with the MWOA algorithm by the parameters of Latency, Execution time, and Memory size. The latency, bandwidth, memory size, and execution time of the model are improved by using the Kafka model with the MWOA optimization algorithm. The information between the producer and the consumer is balanced by using load-balancing optimization algorithms. The main advantage of using Kafka is scalable, low latency, durability, and high concurrency, and the MWOA algorithm are well suitable for balancing the data to reduce the overloading of the information. Hence the combination of the Kafka model using the MWOA optimization algorithm results in efficient performance as the latency of 4000ms, Execution time is 980, and Memory is utilized as 390kb. The table-2 represents the comparative analysis between the existing algorithms to the Modified Whale Optimization algorithm on Kafka.

Table 2: Comparative analysis between various optimization algorithms

Algorithms	Latency(ms)	Execution time(ms)	Memory size(kb)
MPSO [16]	-	1800	640
HHOA [21]	5100	1300	550
ACO algorithm [22]	4900	1207	433
Proposed algorithm	4000	980	390

6. Conclusion

The payload benchmarking with dynamic load balancing for distributed storage systems is analyzed by using the benchmark configuration files as the input data. The input data is processed by using an efficient Kafka method which gives efficient results for fault tolerance and scalability in distributed storage systems and enables the streaming of data in real time. The huge amount of information between the systems is validated by the parameters like latency, throughput, bandwidth, memory size, and execution time. The proposed Kafka model using a modified Whale optimization algorithm results efficient results the latency of 4000ms, execution time of 980ms, and 390kb memory size as compared to the existing methods like a spark, storm, Flink, and optimizations algorithms like PSO, GOA, GWO. Further, the privacy and security of the information are evaluated and the benchmarking data is processed with minimum cost, and energy consumption to result in more efficient load balancing.

References

Xingjun, L., Zhiwei, S., Hongping, C. and Mohammed, B. O. (2020). A new fuzzy-based method for load balancing in the cloud-based Internet of things using a grey wolf optimization algorithm. International Journal of Communication Systems, 33(8), 4370

Sfaxi, L. & Aissa, M. M. B. (2021). Babel: A Generic Benchmarking Platform for Big Data Architectures. Big Data Research, 24, 100186

Barba-González, C., Nebro, A. J., Benítez-Hidalgo, A., García-Nieto, J. & Aldana-Montes, J. F. (2020). On the design of a framework integrating an optimization engine with streaming technologies. Future Generation Computer Systems, 107, 538-550

Brandón, Á., Solé, M., Huélamo, A., Solans, D., Pérez, M.S. and Muntés-Mulero, V. (2020). Graph-based root cause analysis for service-oriented and microservice architectures. Journal of Systems and Software, 159, 110432

Shafiq, D. A., Jhanjhi, N. Z., Abdullah, A. and Alzain, M. A. (2021). A load balancing algorithm for the data centres to optimize cloud computing applications. IEEE Access, 9, 41731-41744

Talaat, F. M., Saraya, M. S., Saleh, A. I., Ali, H. A. & Ali, S. H. (2020). A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment. Journal of Ambient Intelligence and Humanized Computing, 11(11), 4951-4966

Peng, J., Cai, K. and Jin, X. (2020). High concurrency massive data collection algorithm for IoMT applications. Computer Communications, 157, 402-409

Henning, S. and Hasselbring, W. (2021). Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures. Big Data Research, 25, 100209

Devaraj, A. F. S., Elhoseny, M., Dhanasekaran, S., Lydia, E. L. & Shankar, K., (2020). Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments. Journal of Parallel and Distributed Computing, 142, 36-45.

Neelima, P. and Reddy, A., (2020). An efficient load balancing system using adaptive dragonfly algorithm in cloud computing. Cluster Computing, 23(4), 2891-2899

Balaji, K., Kiran, P. S. & Kumar, M. S. (2021). An energy efficient load balancing on cloud computing using adaptive cat swarm optimization. Materials Today: Proceedings

Khriji, S., Benbelgacem, Y., Chéour, R., Houssaini, D. E. & Kanoun, O. (2022). Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. The Journal of Supercomputing, 78(3), 3374-3401

Barba-González, C., Nebro, A. J., Benítez-Hidalgo, A., García-Nieto, J. & Aldana-Montes, J. F. (2020). On the design of a framework integrating an optimization engine with streaming technologies. Future Generation Computer Systems, 107, 538-550

Goyal, S., Bhushan, S., Kumar, Y., Rana, A. U. H. S., Bhutta, M. R., Ijaz, M. F. & Son, Y. (2021). An optimized framework for energy-resource allocation in a cloud environment based on the whale optimization algorithm. Sensors, 21(5), 1583

Chakraborty, S., Saha, A. K., Sharma, S., Mirjalili, S. & Chakraborty, R. (2021). A novel enhanced whale optimization algorithm for global optimization. Computers & Industrial Engineering, 153, 107086

Jena, U. K., Das, P. K. & Kabat, M. R. (2020). Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. Journal of King Saud University-Computer and Information Sciences

Sankaranarayanan, S., Rodrigues, J. J., Sugumaran, V. & Kozlov, S. (2020). Data Flow and Distributed Deep Neural Network based low latency IoT-Edge computation model for big data environment. Engineering Applications of Artificial Intelligence, 94, p.103785

Bordin, M. V., Griebler, D., Mencagli, G., Geyer, C. F. & Fernandes, L. G. L. (2020). DSPBench: A suite of benchmark applications for distributed data stream processing systems. IEEE Access, 8, 222900-222917

Bordin, M. V., Griebler, D., Mencagli, G., Geyer, C. F. & Fernandes, L. G. L. (2020). DSPBench: A suite of benchmark applications for distributed data stream processing systems. IEEE Access, 8, 222900-222917

Li, C., Cai, Q. & Lou, Y. (2022). Optimal data placement strategy considering capacity limitation and load balancing in geographically distributed cloud. Future Generation Computer Systems, 127, 142-159

Annie Poornima Princess, G. & Radhamani, A. S. (2021). A hybrid meta-heuristic for optimal load balancing in cloud computing. Journal of Grid Computing, 19(2), 1-22

Muteeh, A., Sardaraz, M. and Tahir, M., 2021. MrLBA: Multi-resource load balancing algorithm for cloud computing using ant colony optimization. Cluster Computing, 24(4), 3135-3145