European Journal of Molecular & Clinical Medicine
ISSN 2515-8260 Volume 07, Issue 07, 2020

Cross Entropy with Glowworm Swarm Optimization Algorithm based Load Balancing Technique for Distributed Big Data Systems

Gurrampally Kumar[1], Dr. S. Mohan[2]

[1]*Department of Computer Science and Engineering, Lords Institute of Engineering and Technology, Hyderabad, Telangana, India.*

[2]*Department of Computer Science and Engineering, Annamalai University, Annamalai Nagar, Chidambaram, Tamilanadu, India.*

[1]*grk.040@gmail.com*

[2]*mohancseau@gmail.com*

Abstract: ***Presently, digital data gets exponentially raised owing to an increase in number of data channels which generate and distribute data, load balancing techniques are developed for handling big data in real time. Though the cloud environment offers effectual services, it faces some serious issues of load balancing where the improper distribution of load results in degraded overall processing performance. This paper presents a novel Cross Entropy with Glowworm Swarm Optimization Algorithm based Load Balancing (CEGSO-LB) Technique for Distributed Big Data Systems. The aim of the CEGSO-LB model is to reduce the overall processing cost and schedule the load on the VMs proficiently. The presented CEGSO algorithm incorporates the basic concepts of CE method and GSO algorithm. The CE concept is integrated into the GSO algorithm to improve the efficiency in attaining global solutions and eliminating local optima problem. The presented model is implemented to examine the results under varying sizes of synthetic datasets and varying number of Virtual Machines (VMs). The experimental results guaranteed the betterment of the CEGSO-LB technique interms of distinct aspects namely Average Load, Average turnaround time, Average response time, CPU utilization, memory utilization, reliability, average execution time, makespan, and average throughput.***

Keywords: Big data, distributed systems, cloud computing, load balancing, virtual machines

1. INTRODUCTION:

Presently, the exploitation of digital gadgets like tablets, personal computers, smartphones has raised the quick development of social media applications namely Twitter and Facebook. It has resulted in an enormous generation of digital data in day-to-day lives. Consequently, big data technologies have become an effective way of managing and processing huge quantity of data which could not be managed in a classical way [1]. Big data has imposed the latest meaning of processing and examining massive quantity of data which is not available in the existing data. It is commonly employed in distinct domains namely trend analysis, marketing, and decision making [2, 3]. Distributed storage processing models namely Hadoop are available for processing huge quantity of data outside the computational restricts of the available storage

and processing system [4]. Hadoop is an illustrative openly accessible model used for the distributed storage and computation of massive quantity of data. Since Hadoop saves and computes massive data quantity in the disk of distributed nodes, incessant disk input and output happen, resulting in real-time processing incredible [5]. Also, if the inputs and outputs are focused on a particular node, the bottleneck happens the total processing speed gets reduced. Fig. 1 illustrates the structure of dynamic algorithms of load scheduling [6].

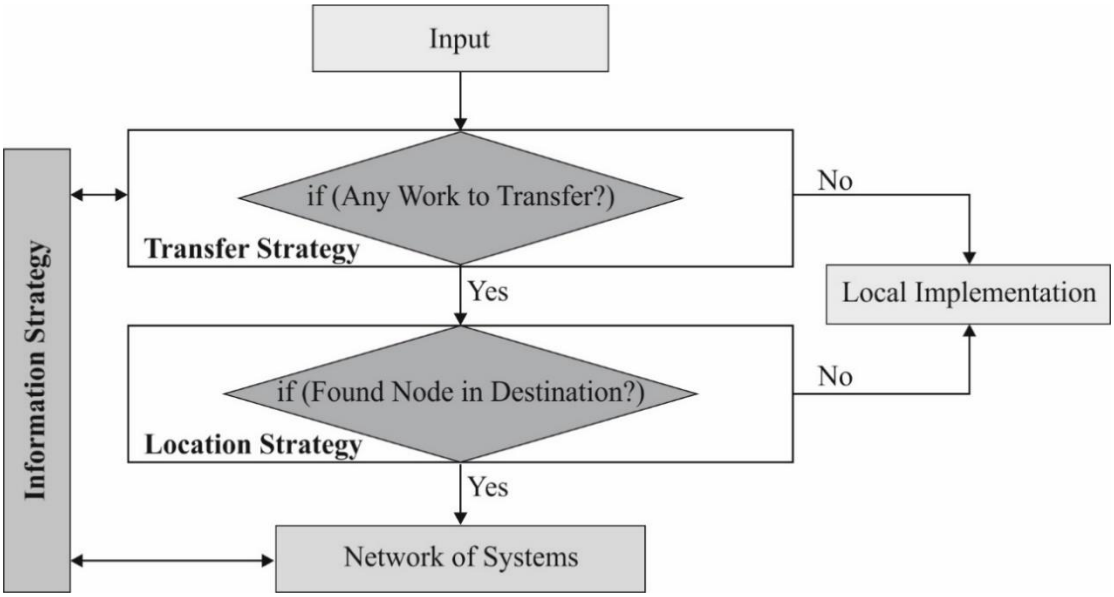

Fig. 1 Structure of Dynamic Algorithms of Load Scheduling

For addressing these disk input and output issues, distributed in-memory technology has developed enabling the way to distribute, store, and process data in the storage to attain quick access speed [7]. Distributed in-memory technologies are commonly utilized for application areas which process massive quantity of data in real time scenarios. A sample representative in-memory processing technique is called Memcached [8]. It is a key based memory caching technique commonly used in application areas offering online real time services like Youtube, Instagram, etc. It minimizes the memory utilization to the backend database by straightaway storing the data request of the user in the distributed in-memory [9]. Since Memcached operates works on distributed environment, load imbalance issue exists between the nodes. Alternatively, in the distributed in-memory environment, if the requests are focused on a particular node or the utilization of specific data is concentrated, the issue of load increase on a particular node takes place. This load imbalance problem amongst the node reduces the total system response time and network efficiency [10].

This paper presents a novel Cross Entropy with Glowworm Swarm Optimization Algorithm based Load Balancing (CEGSO-LB) Technique for Distributed Big Data Systems. The aim of the CEGSO-LB model is to reduce the overall processing cost and schedule the load on the VMs proficiently. The presented CEGSO algorithm incorporates the basic concepts of CE method and GSO algorithm. The CE concept is integrated into the GSO algorithm to improve the efficiency in attaining global solutions and eliminating local optima problem. The presented model is implemented to examine the results under varying sizes of synthetic datasets and varying number of VMs.

2. LITERATURE REVIEW

For addressing the load imbalance problem amongst the nodes in the distributed in-memory environment, several works have been performed by the use of ring based hashing techniques [20-24]. Generally available ring based hashing techniques modify the load by data replication process to other nodes or data migration by a hash space modification. [11] computed the load on the nodes by the use of hit and usage rates, and carried out the LB process by modifying the hash space. When the hot data is not available in the particular node, but, the hit ratio and usage rate get increased and several hash spaces should be modified. Alternatively, the existence of the hot data considerably enhances the cost of data migration. The works in [12] presented a model of load distribution by focusing on only one node by repeating the hot data which results in a large load to another node. If LB is carried out by the consideration of hot data, it can be impossible to resolve the situation where the load appears on the node with no hot data. In addition, in case of heterogeneous platform, LB is carried out without considering the storage space, adequate data migration exists in the node. Therefore, it is tedious to employ the available LB techniques due to the fact that it performs LB for a particular scenario.

In [13], Binary Gravitational Search Algorithm (BGSA) is introduced for optimization of the scheduling operation generated from distinct environment. A hybridization of GSA is presented in [14] by the use of orthogonal crossover as well as patterns search for load scheduling in cloud platforms. Besides, two effective GSA optimization algorithms are presented in [15] for enhancing the particle diversity and employ the storage models in the mathematical computations. It has designed the security parameters based on the behavioral graph and defined the focus on LB and service assignment in cloud platform. Some other recent works are involved in this area. A study in [16] presented a novel PROUD technique for securing the outsource data designcryption process for edge servers to reduce the processing overhead on the client end. In addition, a novel CLoud scientific wOrkflowSchedUlingalgoRithm based on attack–defensE game model (CLOSURE) [17] is presented for scheduling load in cloud environment.

3. THE PROPOSED CEGSO-LB TECHNIQUE

The presented CEGSO-LB technique follows the idea of GSO algorithm and CE mechanism. The aim of the CEGSO-LB model is to reduce the overall processing cost and schedule the load on the VMs proficiently. The processing cost contains the transferring and execution cost of the cloudlets. It provides effective searching area exploitation and user satisfaction by deriving a fitness function. The parameters involved in the fitness function of the cloudlet and VM are MIPS, bandwidth, execution cost, and transfer cost. The cloudlet scheduling at the VMs is performed. The data center has $(VMs)^{loudlets}$ probable methods of executing the cloudlets on the respective VMs. In case of implementing 3 cloudlets on 2 VMs, then the probability becomes 8. The glowworms S undergo initialization at the CloudSim tool as represented as follows:

$$S_i = \left(s_i^1, s_i^2, \dots\dots, s_i^n, \dots\dots\dots, s_i^d\right)$$
$$\forall i = 1 to\ 25\ and\ n = 1\ to\ 10 \quad (1)$$

The fitness function determines the fitness value of the glowworms in the searching area. The initial glowworm uses CE next to the choice of subsequent glowworm by the use of optimum fitness value. Assume a $Ct_{exe}(M)_j$ is the entire execution cost of all glowworms allocated to calculate the VM resources PCj. It is found by summing the weights assigned to the nodes in the mapping of glowworms of all cloudlets allocated to individual resources.

Let $Ct_{fron}(M)_j$ indicates the sum of the transfer cost which existed amongst the cloudlets allocated to calculate the VM resource PCj. The output indicates the product of the output file size and transmission cost. The average cost of data amongst a set of two resources of transmission is defined by $d(k1), S(\mathrm{k2})$ and the glow worms are independent of each other. The overall cost is included for every glowworm by the inclusion of the execution and transfer cost and then the cost is also reduced for estimating the fitness function.

$$\mathrm{Ct}_{exec}(S)_j = \sum_k \omega_{kj}\,, \forall S(\mathrm{k}) = j \tag{2}$$

$$\mathrm{Ct}_{trans}(S)_j = \sum_{\mathrm{k1}\in T}\sum_{\mathrm{k2}\in T} d_{S(\mathrm{k1}),S(\mathrm{k2})} * e_{k1,\mathrm{k2}}, \tag{3}$$

$$\forall S(k1) = j \; and \; S(1\langle 2) \neq j \tag{4}$$

$$\mathrm{Ct}_{\mathrm{total}}(S)_j = \mathrm{Ct}_{exec}(S)_j + \mathrm{Ct}_{trans}(S)_j \tag{5}$$

$$Cost_{Total}(S) = \max\left(\mathrm{Ct}_{\mathrm{total}}(S)_j\right), \forall j \in S \tag{6}$$

$$Minimize\ (\mathrm{Cost}_{Total}(S)\,, \forall S) \tag{7}$$

A. Principle of GSO Algorithm

In the GSO algorithm, a collective set of glowworms undergo initial random deployment in the solution space. Every individual glowworm indicates a solution of objective function in the searching space and holds a particular amount of luciferin in it. The amount of luciferin is linked to the fitness level of the present position of the agent. The brighter level of the glowworm represents an optimal solution. By the use of probability based models, the agents are attracted with the adjacent agents whose luciferin intensity exceeds the own inside the local decision domain and afterward shift toward it. The density of the glowworm's neighbor influences the effect of the decision radius and computes the size of the local decision domain. If the neighboring density is found to be low, then the local decision domain gets enlarged for the identification of several neighbors; else, it reduces the enables of the swarm division into a smaller set of groups. These processes get iterated till the GSO algorithm reaches the stopping criteria. Here, most of the individuals gather over the brighter individuals [18]. In short, a set of 5 major stages are involved in the GSO algorithm namely luciferin update, neighborhood select, moving probability computer, movement, and the decision radius update.

The luciferin update stage is mainly based on the fitness value and earlier luciferin value, and the rule can be represented as follows

$$l(t+1) = (1-\rho)l_i(t) + \gamma\, Fitness\left(x_j(t+1)\right). \tag{8}$$

where $l_i(t)$ represents the luciferin value of glowworm i at time t, ρ is the luciferin decay constant, γ is the luciferin enhancement constant; $x_i(t+1) \in R^M$ is the position of the glowworm i at time $t+1$, and Fitness $(x_j(t+1))$ denotes the fitness value of the glowworm i's position at time $t+1$. Fig. 2 demonstrates the flowchart of GSO technique.

At the neighbor-select stage, the neighbors $N(t)$ of the glowworm i at t time comprises of brighter individuals and is defined by

$$N_i(t) = \left\{j: d_{ij}(t) < r_d^i(t);\ l_i(t) < l_j(t)\right\}. \tag{9}$$

where $d_{ij}(t)$ defines the Euclidean distance among the glowworms i and j at time t, and $r_d^i(t)$ signifies the decision radius of glowworms i at time t.

In the Moving Probability Computer stage, the glowworm utilizes a probability rule for moving in the direction of other glowworms with maximum luciferin level. The probability $P_i(t)$ of glowworm i which moves in the direction of the neighbor, j can be represented by:

$$P_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)}. \quad (10)$$

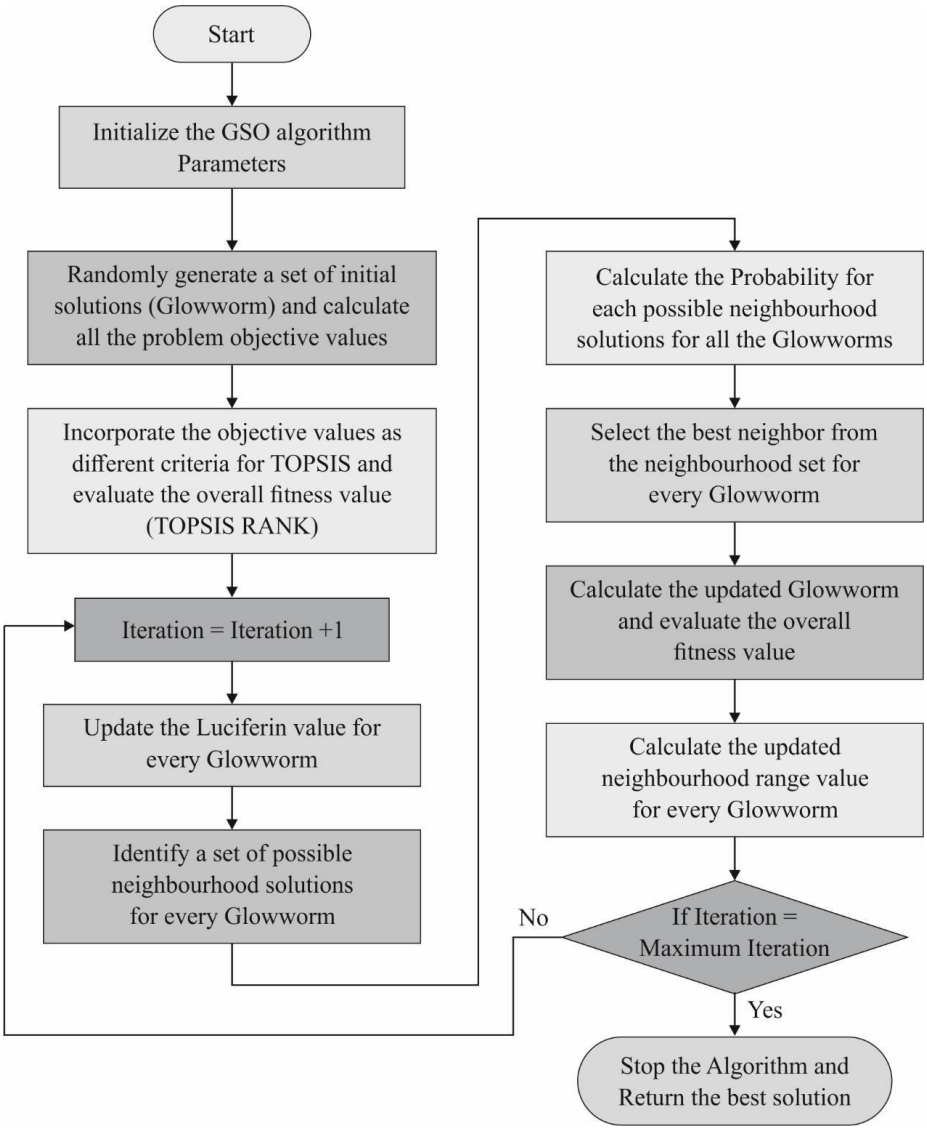

Fig. 2 Flowchart of GSO algorithm

At the movement stage, assume the glowworm i chooses a glowworm $j \in N(t)$ with $P_{ij}(t)$; the discrete time model of the movement of glowworm i can be represented by

$$x(t+1) = x_i(t) + s\left(\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}\right). \quad (11)$$

where, $\|\cdot\|$ defines the Euclidean norm operator, and s denotes the step size. Finally, under the decision radius update stage, the decision radius of the glowworm i can be represented below:

$$r_d^i(t+1) = \min\left\{r_s, \max\left\{0, r_d^i(t) + \beta\left(n_t - \left|N_j(t)\right|\right)\right\}\right\}. \quad (12)$$

where, β is a constant, r_s means the sensory radius of glowworm i, and n_t is a controlling variable of the neighbor number.

B. Concept of Cross Entropy

The CE model considers the benefits of sampling a problem space by producing the candidate solutions by the use of the distribution, then it updates the distribution depending upon the effective candidate solution exposed. The CE operator not only enlarges the searching area; it also ensures that the newly generated solutions are considered from the nearby useful details into account. For improving the effectiveness of the classical GSO algorithm to find the optimal solution and avoid the local optimum problem, the CEGSO algorithm is developed. Here, CE model is used for updating the members and fix a Time-to-Live (TtL) parameter for resolving the local optimum issue. The CE technique for optimization is defined below.

$$(x^*) = \gamma^* = \max_{x \in X} S(x), \tag{13}$$

where γ^* is the maximum over the assumed set X, the x^* is the utmost x. S is the evaluation parameter. If the determined sample instances X in an iterative way, a collection of indicator functions $_{\{S(X)\geq\gamma\}}$ are represented. $_{\{S(X)\geq\gamma\}}$ signifies the $S(x)$ as above in the level γ for sample instance x. For a vector u, m of probability density function variables, the optimization issue is converted by the estimation of the probability $P(S(X) \geq \gamma)$. In the process of integrating with the indicator functions, the probability can be determined as:

$$l(\gamma) = P_u(S(X) \geq \gamma) = \sum_x I_{\{S(X)\geq\gamma\}} f(x,u) = \mathrm{E}_u I_{\{S(X)\geq\gamma\}}, \tag{14}$$

where P is the probability linked to the probability density function $f(.,u)$, and E_u defines the expectation function. When $\gamma = \gamma^*$, (γ) is determined by

$$argmax\ \frac{1}{N}\sum_{i=1}^{N}\{S(X) \geq \gamma\}\, lnf(X_i, v). \tag{15}$$

X_i is produced by the use of pdf $(.,v)$. It is well-intentioned to notice that the CE technique determines the well sampling density $(.,v^*)$ in such a way that the optimum solutions undergo sampling [19]. The process of CE is divided into 3 different steps as given below.

1. Generation of arbitrary sample instances takes place from Gaussian distribution with mean *mu* and standard deviation s.
2. Choose a certain number of optimal sample instances from the entire sample instances.
3. Updates *mu* and s depending upon the optimal sample instances with efficient fitness.

Algorithm 1: Steps involved in CE method
Initialize: mean mu, standard deviation s, size of population pop, the number of best samples np, terminal condition t_{max} While $t < t_{max}$ do Create sample vectors X as: $x_j = mu + s \times randn()x_i$, where randn( ) generates a Gaussian distribution random number. Assess x_i Choose the np optimal sample instances from pop Update the variables mu, s

C. Application of CEGSO algorithm for Load Scheduling

As discussed earlier, the CEGSO-LB model integrates the concepts of GSO and CE, the evaluation function is defined in Eq. (16):

$$Evaluation\ function = 1 - \left(\alpha_{ti} \times \frac{t_i - t_{min}}{t_{max} - t_{min}} + \alpha_{ci} \times \frac{c_i - c_{min}}{c_{max} - c_{min}} \right) \quad (16)$$

At this point, each executable process is evaluated to select the optimum order of execution. For proficient scheduling of resources, the presented CEGSO-LB technique is applied. In addition, distinct tasks are considered as input comprising two parameters namely arrival time and run time. At last, the evaluation function is computed by:

$$1 - \left(\alpha_{ti} \times \frac{t_i - t_{min}}{t_{max} - t_{min}} + \alpha_{ci} \times \frac{c_i - c_{min}}{c_{max} - c_{min}} \right) \quad (17)$$

where f_{min} and f_{max} indicates minimum and maximum run time, c_{min} and c_{max} defines the minimum and maximum input time respectively.

4. PERFORMANCE VALIDATION

The performance of the CEGSO-LB model has been validated under varying synthetic datasets and varying VM instances. A set of four synthetic datasets is used where the extra-large has 800-1000 tasks with the size of 100000-200000MI, the large size task has 600-700 tasks with the size of 70000-100000MI, the medium size task has 400-500 tasks with the size of 50000- 70000MI, and the small sized tasks has 100-200 tasks with the size of 30000-50000MI. In addition, the extra-large VM instances have a CPU capacity of 35000MIPS with the memory capacity of 20GB, the large VM instances have a CPU capacity of 25000MIPS with a memory capacity of 15GB, the medium VM instances has a CPU capacity of 20000MIPS with the memory capacity of 10GB, and the small VM instances has a CPU capacity of 10000MIPS with the memory capacity of 5GB respectively.

A detailed comparative results analysis is made with the existing techniques such as Random Deployment (RD), Weighted Round Robin (WRR), Dynamic Load Balancing (DLB), Load Balancing based on Bayes and Clustering (LB-BC), and Load Balancing Resource Clustering (LBRC).

Fig. 3 shows the results analysis of the CEGSO-LB model with other scheduling techniques interms of average load. The figure demonstrated that the SJF algorithm has resulted in an insignificant outcome by offering a maximum average load of 0.495ms. Likewise, the FireFly (FF) technique has obtained a slightly improved average load of 0.47ms whereas the FCFS algorithm has reached an even better average load of 0.46ms. Besides, the IPSO algorithm has tried to show moderate outcomes with an average load of 0.457ms whereas even better average load of 0.43ms is attained by the RR algorithm. Followed by, the GA has depicted somewhat reasonable outcome with an average load of 0.31ms whereas the IPSO-FF algorithm has accomplished a certainly acceptable average load of 0.259ms. Though the FIMPSO algorithm has exhibited near optimal average load of 0.247ms, the presented CEGSO-LB model has demonstrated effective performance by providing a least average load of 0.198ms.

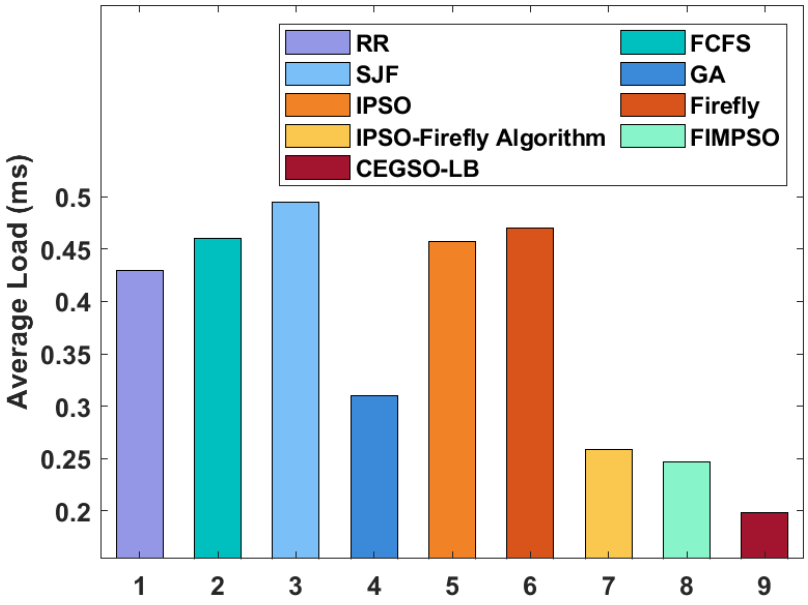

Fig. 3 Average load analysis of CEGSO-LB model

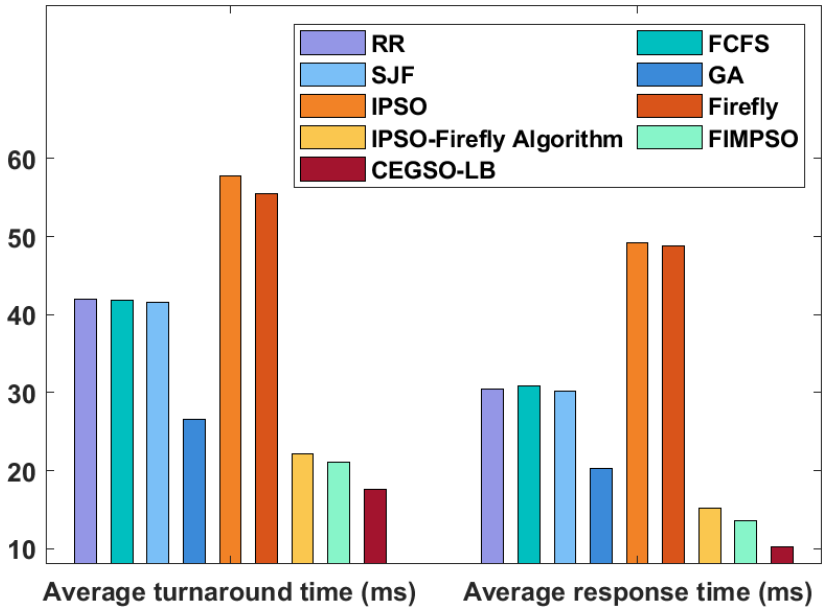

Fig. 4 ATT and ART analysis of CEGSO-LB model

Fig. 4 displays the comparative Average Turnaround Time (ATT) and Average Response Time (ART) analysis of the CEGSO-LB with existing methods. The figure showcased that the IPSO algorithm has resulted in an insignificant result by offering a higher ATT and ART of 57.74ms and 49.23ms. Likewise, the FF technique has obtained a slightly reduced ATT and ART of 55.54ms and 48.87ms whereas the RR approach has reached an even better ATT and ART of 41.98ms and 30.50ms. Also, the FCFS method has tried to illustrate moderate outcomes with the ATT and ART of 41.87ms and 30.84ms whereas even better ATT and ART of 41.56ms and 30.24ms is reached by the SJF technique. At the same time, the GA has portrayed somewhat reasonable outcome with the ATT and ART of 26.57ms and 20.30ms whereas the IPSO-FF methodology has accomplished a certainly acceptable ATT and ART of 22.13ms and 15.21ms. Though the FIMPSO algorithm has exhibited near optimal ATT and

ART of 21.09ms and 13.58ms, the proposed CEGSO-LB model has outperformed effective performance by providing a least ATT and ART of 17.56ms and 10.28ms.

Fig. 5 demonstrates the CPU utilization analysis of the CEGSO-LB model with other existing algorithms under varying types of tasks. From the figure, it is evident that the RD algorithm has shown ineffective performance by offering the least CPU utilization of 59% whereas the WRR and DLB algorithms have depicted slightly improved CPU utilization of 63.5% and 68% respectively. In line with this, the LB-RC model has tried to exhibit moderate CPU utilization of 73.25%. Concurrently, the LB-RC, IPSO-FF, and FIMPSO algorithms have attained reasonable CPU utilization of 80.5%, 82.5%, and 84.5% respectively. But the presented CEGSO-LB model has resulted in an effective CPU utilization of 87.75%.

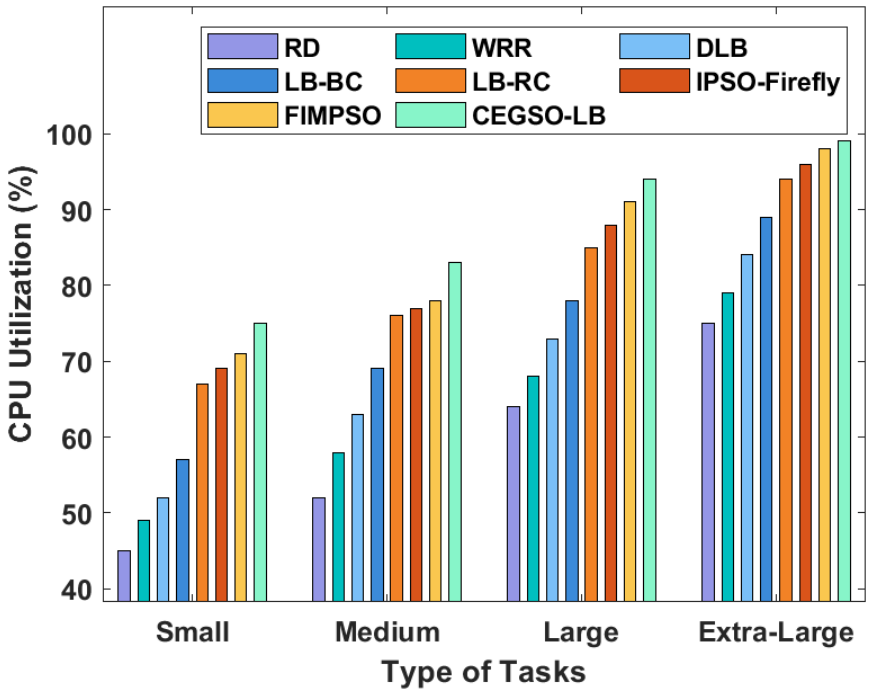

Fig. 5 CPU utilization analysis of CEGSO-LB model

Fig. 6 illustrates the memory utilization analysis of the CEGSO-LB method with other existing techniques under varying types of tasks. From the figure, it is stated that the RD algorithm has depicted ineffective performance by offering the least memory utilization of 55.75% whereas the WRR and DLB approaches have showcased somewhat higher memory utilization of 59.5% and 64.5% correspondingly. Similarly, the LB-RC model has tried to show moderate memory utilization of 68%. Also, the LB-RC, IPSO-FF, and FIMPSO methods have attained reasonable memory utilization of 73%, 75.25%, and 77% respectively. However, the proposed CEGSO- LB model has resulted in an effective memory utilization of 82.75%.

Fig. 7 exhibits the reliability analysis of the CEGSO-LB model with other existing methods under varying types of tasks. From the figure, it is revealed that the RD algorithm has demonstrated ineffective performance by offering minimum reliability of 55% whereas the WRR and DLB models have depicted slightly superior reliability of 60% and 66% respectively. In addition, the LB-RC model has tried to portray moderate reliability of 73.5%. Concurrently, the LB-RC, IPSO-FF, and FIMPSO algorithms have attained reasonable reliability of 81.25%, 82.5%, and 84.75% correspondingly. At last, the presented CEGSO-LB model has resulted in effective reliability of 90.75%.

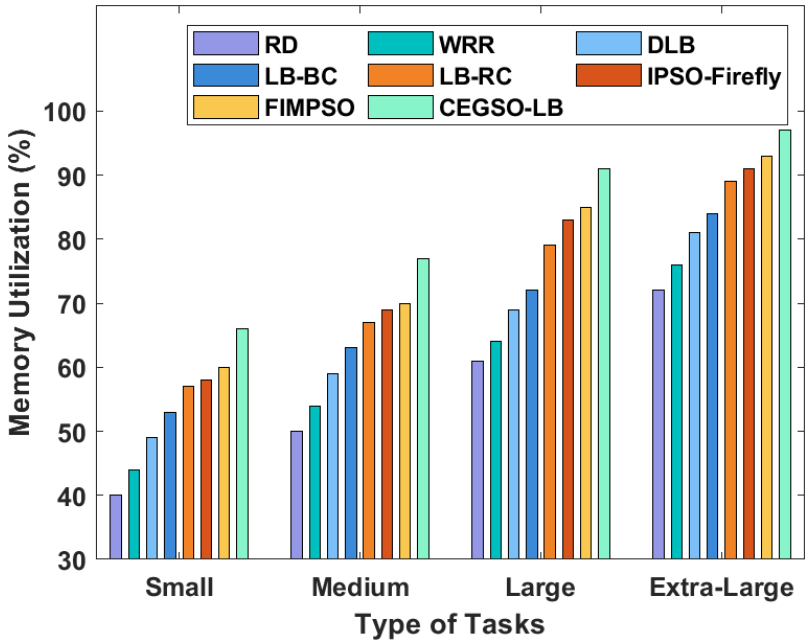

Fig. 6 Memory utilization analysis of CEGSO-LB model

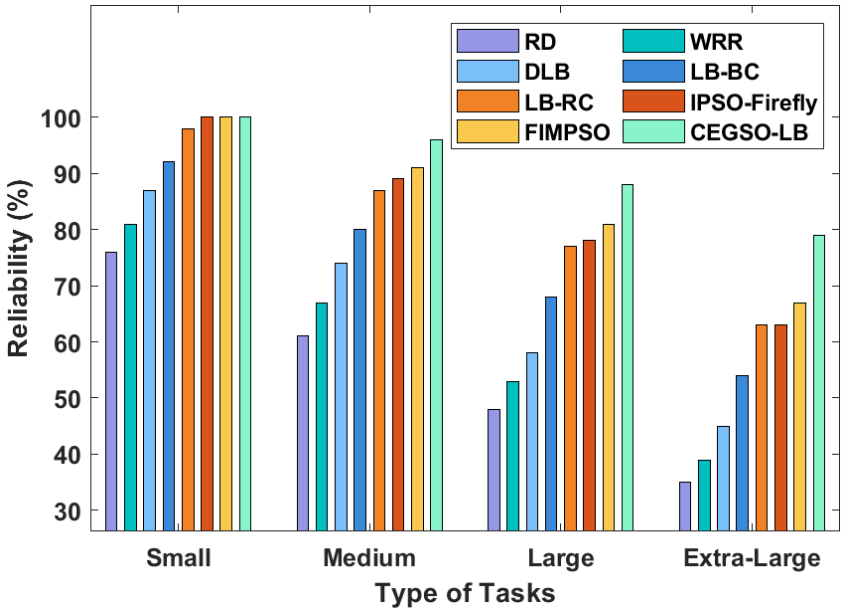

Fig. 7 Reliability load analysis of CEGSO-LB model

Fig. 8 showcases the average execution time analysis of the CEGSO-LB method with other existing models under varying types of tasks. The figure has shown that the RD algorithm has resulted in an insignificant result by offering a maximum average execution time of 71ms. Simultaneously, the WRR approach has obtained a somewhat reduced average execution time of 65.25ms whereas the DLB technique has attained an even better average execution time of 60.25ms. Besides, the LB-BC approach has tried to show moderate outcomes with an average execution time of 53ms whereas even better average execution time of 48.25ms is obtained by the LB-RC algorithm. Followed by, the IPSO-FF has depicted somewhat reasonable outcome with an average execution time of 46.5ms whereas the FIMPSO algorithm has accomplished a

certain acceptable average execution time of 43.75ms. Though, the proposed CEGSO-LB model has portrayed effective performance by providing a minimum average execution time of 38.75ms.

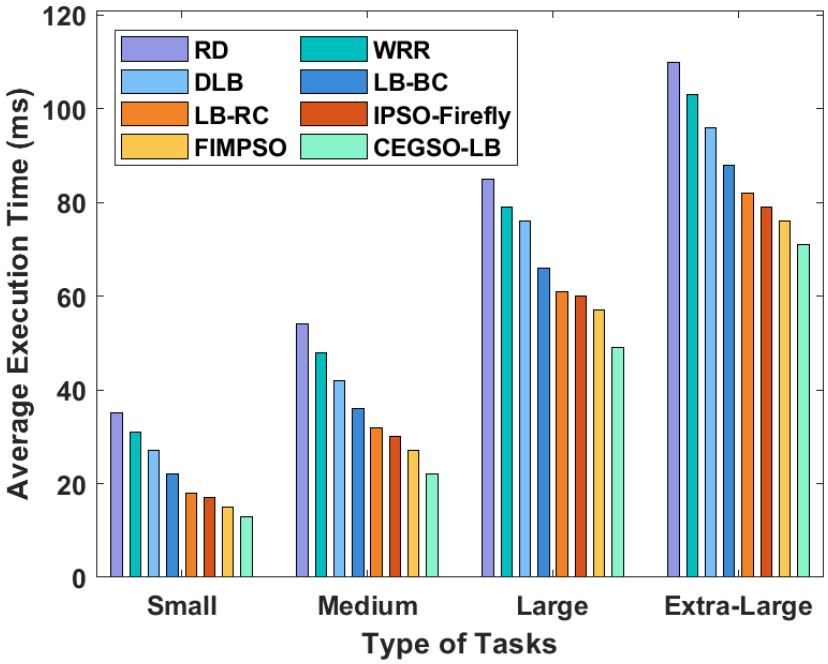

Fig. 8 Average execution time analysis of CEGSO-LB model

Fig. 9 depicts the makespan analysis of the CEGSO-LB technique with other existing algorithms under varying types of tasks. The figure exhibited that the RD methodology has resulted in an insignificant result by offering a superior makespan of 168.5. At the same time, the WRR algorithm has achieved a slightly reduced makespan of 161.25 whereas the DLB algorithm has reached an even better makespan of 154.75. Besides, the LB-BC algorithm has tried to illustrate moderate outcome with the makespan of 147.75 whereas even better makespan of 117 is attained by the LB-RC algorithm. Additionally, the IPSO-FF has depicted somewhat reasonable outcome with the makespan of 114.5 whereas the FIMPSO algorithm has accomplished a certainly acceptable makespan of 112. Eventually, the projected CEGSO-LB technique has showcased effective performance by providing a minimal makespan of 105.5.

Fig. 10 showcases the average throughput analysis of the CEGSO-LB model with other existing techniques under varying types of tasks. From the figure, it is represented that the RD model has depicted ineffective performance by offering a lesser average throughput of 47.75% whereas the WRR and DLB methodologies have depicted slightly higher average throughput of 54.5% and 62.25% correspondingly. Along with that, the LB-RC model has tried to exhibit moderate average throughput of 70.75%. Similarly, the LB-RC, IPSO-FF, and FIMPSO algorithms have achieved reasonable average throughput of 79.25%, 80.25%, and 83.5% correspondingly. However, the presented CEGSO-LB method has resulted in an effective average throughput of 89.75%.

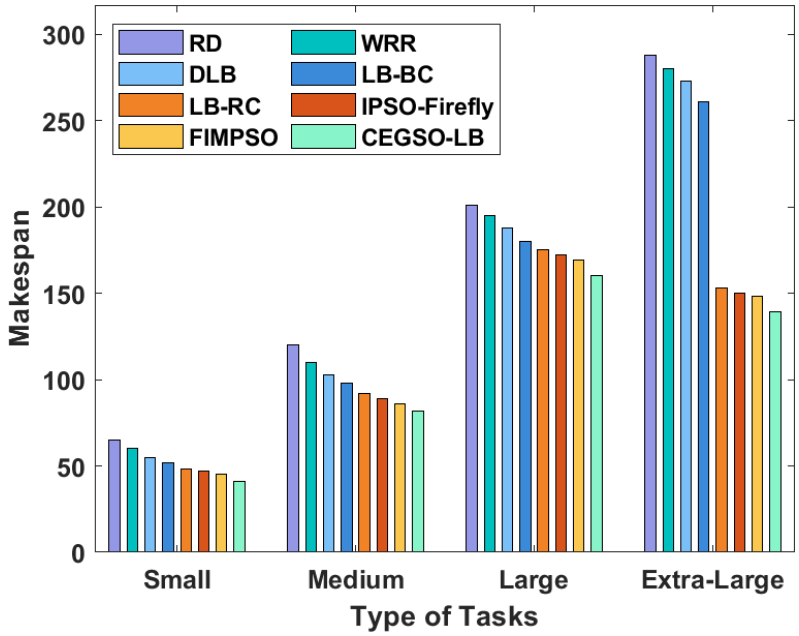

Fig. 9 Makespan analysis of CEGSO-LB model

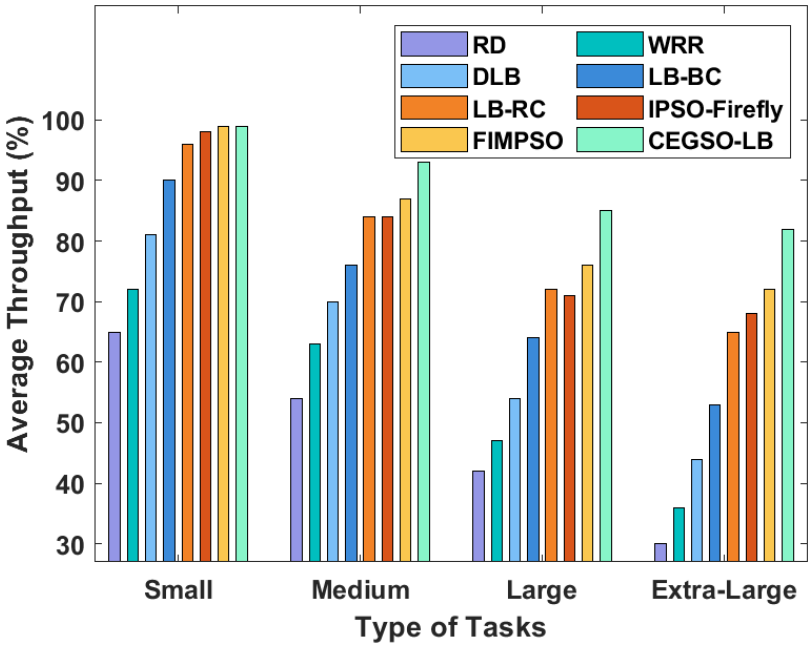

Fig. 20 Average throughput analysis of CEGSO-LB method

5. CONCLUSIONS

This paper has presented an efficient CEGSO-LB Technique for Distributed Big Data Systems. The goal of the CEGSO-LB model is to reduce the overall processing cost and schedule the load on the VMs proficiently. For improving the effectiveness of the classical GSO algorithm to find the optimal solution and avoid the local optimum problem, the CEGSO algorithm is developed. Here, CE model is used for updating the members and fix a TtL parameter for resolving the local optimum issue. The overall cost is included for every

glowworm by the inclusion of the execution and transfer cost and then the cost is also minimized to estimate the fitness function. The presented model is implemented to examine the results under varying sizes of synthetic datasets and varying number of VMs. The experimental results guaranteed the betterment of the CEGSO-LB technique interms of distinct aspects namely Average Load, ATT, ART, CPU utilization, memory utilization, reliability, average execution time, makespan, and average throughput.

6. REFERENCES

[1] Gandomi, A.; Haider, M. Beyond the hype: Big data concepts, methods, and analytics. Int. J. Inf. Manag. 2015, 35, 137–144.
[2] Oussous, A.; Benjelloun, F.Z.; Lahcen, A.A.; Belfkih, S. Big Data technologies: A survey. J. King Saud Univ.-Comput. Inf. Sci. 2018, 30, 431–448.
[3] Lv, Z.; Song, H.; Basanta-Val, P.; Steed, A.; Jo, M. Next-Generation Big Data Analytics: State of the Art, Challenges, and Future Research Topics. IEEE Trans. Ind. Inform. 2017, 13, 1891–1899.
[4] Azzedin, F. Towards a scalable HDFS architecture. In Proceedings of the International Conference on Collaboration Technologies and Systems, San Diego, CA, USA, 20–24 May 2013; pp. 155–161.
[5] Inoubli, W.; Aridhi, S.; Mezni, H.; Maddouri, M.; Nguifo, E.M. An experimental survey on big data frameworks. Future Gener. Comput. Syst. 2018, 86, 546–564.
[6] Golchi, M.M., Saraeian, S. and Heydari, M., 2019. A hybrid of firefly and improved particle swarm optimization algorithms for load balancing in cloud environments: Performance evaluation. Computer Networks, 162, p.106860.
[7] Tiwari, D.; Solihin, Y. MapReuse: Reusing Computation in an In-Memory MapReduce System. In Proceedings of the International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, 19–23 May 2014; pp. 61–71.
[8] Cheng, W.; Ren, F.; Jiang, W.; Zhang, T. Modeling and Analyzing Latency in the Memcached system. In Proceedings of the International Conference on Distributed Computing Systems, Atlanta, GA, USA, 5–8 June 2017; pp. 538–548.
[9] Nishtala, R.; Fugal, H.; Grimm, S.; Kwiatkowski, M.; Lee, H.; Li, H.C.; McElroy, R.; Paleczny, M.; Peek, D.; Saab, P.; et al. Scaling Memcache at Facebook. In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation, Lombard, IL, USA, 2–5 April 2013; pp. 385–398.
[10] Jyothi, M. Effective Load Balancing Technique and Memory Management in Cloud. Int. J. Res. Comput. Commun. Technol. 2014, 3, 1246–1251.
[11] Hwang, J.; Wood, T. Adaptive performance-aware distributed memory caching. In Proceedings of the International Conference on Autonomic Computing, San Jose, CA, USA, 26–28 June 2013; pp. 33–43.
[12] Lu, Y.; Sun, H.; Wang, X.; Liu, X. R-Memcached: A consistent cache replication scheme with Memcached. In Proceedings of the Middleware Posters & Demos Session, Bordeaux, France, 8–12 December 2014; pp. 29–30.
[13] E. Rashedi, A. Zarezadeh, Noise filtering in ultrasound images using Gravitational Search Algorithm, in: Iranian Conference on Intelligent Systems, ICIS, 2014.
[14] M. Khatibinia, S. Khosravi, A hybrid approach based on an improved gravitational search algorithm and orthogonal crossover for optimal shape design of concrete gravity dams, Appl. Soft Comput. J. 16 (2014) 223–233.

[15] G. Sun, A. Zhang, X. Jia, X. Li, S. Ji, Z. Wang, DMMOGSA: Diversity-enhanced and memory-based multiobjective gravitational search algorithm, Inform. Sci. 363 (2016) 52–71.
[16] S. Belguith, N. Kaaniche, M. Hammoudeh, T. Dargahi, PROUD: verifiable privacy-preserving outsourced attribute based signcryption supporting access policy update for cloud assisted IoT applications, Future Gener. Comput. Syst. (2019).
[17] Y. Wang, Y. Guo, Z. Guo, T. Baker, W. Liu, CLOSURE: A cloud scientific workflow scheduling algorithm based on attack–defense game model, Future Gener. Comput. Syst. (2019).
[18] Li, Z. and Huang, X., 2016. Glowworm swarm optimization and its application to blind signal separation. Mathematical Problems in Engineering, 2016.
[19] Tang, R., Fong, S., Dey, N., Wong, R.K. and Mohammed, S., 2017. Cross entropy method based hybridization of dynamic group optimization algorithm. Entropy, 19(10), p.533.
[20] A. Muthumari, J. Banumathi, S. Rajasekaran, P. Vijayakarthik, K. Shankar et al., "High security for de-duplicated big data using optimal simon cipher," Computers, Materials & Continua, vol. 67, no.2, pp. 1863–1879, 2021.
[21] Lakshmanaprabu, S.K., Shankar, K., Ilayaraja, M. et al. Random forest for big data classification in the internet of things using optimal features. Int. J. Mach. Learn. & Cyber. 10, 2609–2618 (2019). https://doi.org/10.1007/s13042-018-00916-z
[22] Lakshmanaprabu, S. K., Shankar, K., Khanna, A., Gupta, D., Rodrigues, J. J., Pinheiro, P. R., & De Albuquerque, V. H. C. (2018). Effective features to classify big data using social internet of things. IEEE access, 6, 24196-24204.
[23] Lydia, E. L., Moses, G. J., Varadarajan, V., Nonyelu, F., Maseleno, A., Perumal, E., & Shankar, K. (2020). CLUSTERING AND INDEXING OF MULTIPLE DOCUMENTS USING FEATURE EXTRACTION THROUGH APACHE HADOOP ON BIG DATA. Malaysian Journal of Computer Science, 108-123.
[24] Lakshmanaprabu, S. K., Shankar, K., Rani, S. S., Abdulhay, E., Arunkumar, N., Ramirez, G., & Uthayakumar, J. (2019). An effect of big data technology with ant colony optimization based routing in vehicular ad hoc networks: Towards smart cities. Journal of cleaner production, 217, 584-593.