Hindawi
Scientific Programming
Volume 2018, Article ID 2724692, 16 pages
https://doi.org/10.1155/2018/2724692

Research Article

Multiobjective Glowworm Swarm Optimization-Based Dynamic Replication Algorithm for Real-Time Distributed Databases

Saadi Hamad Thalij[1] **and Veli Hakkoymaz**[2]

[1]*PhD Student, Yildiz Technical University, Dept. of Computer Engineering, Istanbul, Turkey*
[2]*Assoc. Prof., Yildiz Technical University, Dept. of Computer Engineering, Istanbul, Turkey*

Correspondence should be addressed to Saadi Hamad Thalij; saadi.hamad@yahoo.com

Received 24 July 2018; Revised 30 October 2018; Accepted 18 November 2018; Published 4 December 2018

Academic Editor: Autilia Vitiello

Distributed systems offer resources to be accessed geographically for large-scale data requests of different users. In many cases, replication of the vital data files and storing their replica in multiple locations accessible to the requesting clients is vital in improving the data availability, reliability, security, and reduction of the execution time. It is important that real-time distributed databases maintain the consistency constraints and also guarantee the time constraints required by the client requests. However, when the size of the distributed system increases, the user access time also tends to increase, which in turn increases the vitality of the replica placement. Thus, the primary issues that emerge are deciding upon an optimal replication number and identifying perfect locations to store the replicated data. These open challenges have been considered in this study, which turns to develop a dynamic data replication algorithm for real-time distributed databases using a multiobjective glowworm swarm optimization (MGSO) strategy. The proposed algorithm adapts the random patterns of the read-write requests and employs a dynamic window mechanism for replication. It also models the replica number and placement problem as a multiobjective optimization problem and utilizes MGSO for resolving it. The cost models are presented to ensure the time constraint satisfaction in servicing user requests. The performance of the MGSO dynamic data replication algorithm has been studied using competitive analysis, and the results show the efficiency of the proposed algorithm for the distributed databases.

1. Introduction

A distributed system is a group of independent computers that appear to the clients of the system as a solitary PC [1]. In these systems, data objects are placed onto servers which are situated at geologically distant locations. In the recent past, distributed databases have turned into a vital piece of business processing. Maintaining data authentically and offering precise and convenient processing of database queries and updating over numerous locales have been a critical factor in empowering organizations to use data in a range of various locations on a worldwide scale [2]. Standardisation of query languages, and of the relational and object models, has helped in the coordination of various database systems to frame systems of integrated data services [3]. The issues such as guaranteeing the integrity of data, which updates are opportune, and clients get a uniform rate of response regardless of their position in the system they have become real difficulties to database vendors and clients [4].

Replication [5] is the procedure of sharing information in order to guarantee consistency between redundant resources, such as software or hardware components, in order to enhance dependability, fault tolerance, or accessibility. It could either be data replication if similar data are stored on various storage devices or computation replication if a similar task is executed ordinarily. It is the procedure of consequently dispersing duplicates of data and database objects among SQL server instances and keeping the distributed information synchronized [6, 7]. The data replication issue is an extension of the classical file allocation issue. The protected sharing of information in this sort of condition is an unpredictable issue. The proprietors of the diverse data sources often have distinctive approaches on access and the dissemination of the data that they hold [8]. There are two principle sorts of replication conventions: dynamic replication, in which all replicas form simultaneously all input messages, and passive replication, in which only one of the replica forms all input messages and occasionally transmits its present state to alternate replicas so as to maintain consistency [9]. The principle goal of data replication in distributed systems is to expand dependability and to enhance execution of the entire system. The important concern in data replication is the number of replicas that ought to be made and their positioning in order to meet a specific performance goal, which is termed as total operation cost (TOC).

Data distribution and replication offer chances for enhancing execution through parallel query execution, load adjusting, and also through expansion of accessibility of data [10]. In a distributed database system, data are frequently replicated to enhance reliability and availability, thus expanding its dependability. Likewise, data are additionally stored on PCs, where it is most often possibly required in order to minimize the cost of expensive remote access [11]. Decision-making includes handling or applying information and knowledge, and the appropriate information/knowledge mix relies upon the attributes of the decision-making setting [12, 13]. The replication or relocation of shared data get obstructed at subjective areas on chip which require the utilization of the index or broadcast-based mechanisms for query and intelligence authorization, as each piece is likely going to have diverse placement requirements [14, 15]. Replicating the stock and client-related data at these diverse areas is desirable since it gives quick access to the nearby replica and survive disaster cases where all machines of a physical area crash [16]. In any case, the current systems for replication of distributed databases do not give anticipated outcomes due to the failure in choosing the number of replica and the best location for the replica placement [17]. These issues form the basic motivation for this proposed model. In order to tackle these challenges, multiobjective glowworm swarm optimization- (MGSO-) based dynamic data replication algorithm for distributed database systems has been proposed in this study. The remainder of this article is organized as follows. Section 2 provides the literature review of some of the related works. Section 3 describes the CAP theorem and its limitations. Section 4 explains the proposed replication algorithms, while Section 5 provides the simulation results of this model. Section 6 presents the conclusion related to the proposed algorithm.

2. Related Works

Replication is an important part in all distributed systems. The two widely used replication procedures include the following: static replication [18] and dynamic replication [19]. Various researchers have developed replication strategies based on these two procedures. The number of replicas for each file is fixed manually in static replication which is preselected, while in dynamic replication it is determined by an automatic decision system that decides on the number during execution. These decision modules decide on the replica number with consideration of the system infrastructure and also the environments and client's access strategies. In general cloud applications, the dynamic strategies are much preferred as it suits the flexible requirements of the clients.

Khanli et al. [20] developed the predictive hierarchical fast spread (PHFS) model for dynamic replication which decreases the latency in the multitier grid systems. This model is an enhanced version of the previously utilized fast spread [21] and aims at flexibly utilizing the spatial positions [21, 22] of the hosts to serve the user requirements. Though this model reduces the delay and increases the performance in locality access patterns, this model does not consider storage constraints during priority evaluation. Also this model has low adaptability in determining suitable thresholds and time interval for feature varying applications. Previously, Kunszt et al. [23] presented a file-based replication management system that incorporates features required for an application programmer and an end user effectively. However, similar to [20], this model is also aimed at serving multilevel grids with much higher storage, and hence, its suitability to storage constrained systems is very low. Lin et al. [24] introduced eStor, data center storage for ensuring energy-efficient data replication. This model enabled the underutilized storage servers to power off for limiting the energy wastage during the idle state. However, this model did not accompany the predictability evaluation function, and hence, the future clients' requests may face extended delay. Chang et al. [25] developed a dynamic data replication mechanism named latest access largest weight (LALW) suitable for the grid cluster systems. This mechanism selects a popular file from the database and replicates it to suitable number of systems or grids to ensure effective data access. This model also enhances the load balancing or replicas with a registration process that selects the weights based on the time of the requests. The major limitation in this model is the high job execution time as the time interval is short and not suitable selection of based of exponential decay causing single point of failure. Dong et al. [26] introduced a replication strategy over multiple data centers to limit power utilization in the backbone network. However, this work focuses around replication methodologies between various data centers, albeit not inside data centers.

Ping et al. [27] developed the optimal data replication model that operates along the data centers to limit the data access delay in the client side. This model utilized weighted k-means clustering of user locations to determine the optimal replica. However, this model does not utilize the concept of adapting to the varying applications and its storage constraints. Li et al. [28] developed a novel cost-effective dynamic data replication strategy based on the incremental replication method. This strategy reduces the storage cost along with satisfying the data reliability requirements. However, this strategy is efficient only when the data reliability is low and the storage unit aging process is very critical. Also the trade-off between cost and performance is not satisfactory. Perez et al. [11] introduced a replication technique known as the Branch Replication Scheme (BRS) for heterogeneous grid systems. This BRS provides optimized storage usage, increased data access performance, and ability to modify the replicas. These advantages enable the operations like reading or updating a replica in a more efficient manner and reduce the overhead in data access processing. However, BRS is suitable only for the complex applications with dedicated storage space for each branch which is highly unlikely in real-time grids. Similar to BRS, CDRM (cost-effective dynamic replication management) has been proposed for heterogeneous cloud systems in [29]. This model analyses the relationship between availability and replica number and then determines the minimal replica number for given availability. This model is also flexible in adjusting the replica number and location based on the varying workload conditions and node capacities. However, this model has the limitation of delay in analysing the relationship function. Qu and Xiong [30] developed the resilient, fault tolerant, and high efficient (RFH) replication algorithm resolving the flash crowd problem in distributed systems. However, this model has less consistency maintenance which reduces the reliability of its performance.

Lin et al. [31] proposed another approach of utilizing the priority list-based concept of optimal replica placement. This strategy resolved the issues of users' limited authority in resource access, satisfying the spatial requirements during placement stage. This model places the replica in the optical manner such that the workload is balanced through determination of the minimum replica number when the workload capacity is maximized. However, this model has the limitations of resolving the replica placement problem for general graphs and planar graphs. Similarly, Andronikou et al. [32] developed a dynamic QoS-aware replication strategy that utilizes the concept of data importance to select the replica locations. This model analysed the complete lifecycle of the replication, and then, the new replica and generated to migrate the old replica from their current location. This model utilized a set of interoperable novel file replication algorithms to analyse and determine the infrastructure constraints to simplify the replication process based on data popularity. However, this model does not consider all the QoS factors of both service provider and client-related requirements in the market-related constraints. Bonvin et al. [33] introduced a self-organized, fault tolerant, and scalable replication scheme which operates based on the multiple differentiated availability guarantees of the replica to each application. This scheme employed a virtual economy model as a game-theoretical model to determine the optimizer for data partition and mitigate the replica to their respective locations. However, this model suffers from the limitation of data size, i.e., it is much suitable only for smaller data applications. Boru et al. [34] developed an energy efficient data replication strategy by considering both energy consumption and bandwidth consumption. This model reduces the energy bottleneck problem and also reduces the network delays and bandwidth wastage. The only shortcoming of this model is the practical difficulties in implementing it in real-world applications. Mansouri et al. [35] presented a cost optimization model for ensuring dynamic replication and subsequent migration in cloud. This model utilized an optimal offline algorithm of dynamic and linear programming to analyse the workload of the system. Then, two online algorithms are introduced to minimize trade-off between the storage and migration costs followed by the dynamic selection of storage classes. This model reduces the time complexity of the offline algorithms, but the major drawback is its inability to tackle the availability issues. Nagarajan and Mohamed [36] introduced a prediction-based dynamic replication strategy utilizing multiple parameters of the neighboring sites. The parameters considered are storage capacity, bandwidth, and communication cost for selecting and placing the replica. This model also used modified a priori algorithm for predicting the future needs in the grids, but it does not focus on effective scheduling strategies which is highly not recommended. Pan et al. [37] proposed a dynamic replication management strategy based on the load balance condition of the distribution system. This approach improved the replica management, although it considers only the load balance state while ignoring the scheduling of waiting jobs. Though there has been extensive research providing numerous replication strategies, there still exists issues such as optimal selection of replica number and location to store them still continue to remain challenging. This study provides an approach that resolves these issues perfectly than most existing approaches.

3. CAP Theorem

CAP theorem has been displayed in the context of a web service, presented as trade-off between consistency, availability, and partition tolerance [38]. The CAP theorem is a fundamental part of the hypothesis of distributed systems. It expresses that, within the sight of partitions (i.e., network failures), it is not possible for a system to be both consistent and available, and therefore, it becomes important to select one of the two. It has really changed the landscape of how distributed storage systems were architected. It can be expressed as "In a network subject to communication failures, it is impossible for any web service to implement an atomic read/write shared memory that ensures a response to each request." The three fundamental properties of CAP theorem are consistency, availability, and partition tolerance [39].

CAP offers architecture improvement of distribution systems. However, one cannot build up a distributed database system that is continually available, sequentially consistent, and tolerant to partition pattern. It must be assembled only with two of these three properties. Furthermore, certain restrictions degrade the efficiency of CAP theorem [40, 41]. The limitations of CAP are given in [41, 42] as illustrated below:

(i) Binary existence is convenient for proof purposes, but does not closely match the intuitive notion of availability. The traditional CAP theorem's definition does not take into account a quantitative measure of network latency. According to the availability property, if the response has not arrived, there is still hope that the response will arrive, but still it does not have an upper bound on latency.

(ii) Availability requires that only the nonfailed nodes respond. In a networked partitioned area, even if one node fails at a given time, the availability of a system is hampered. In order to ensure complete availability, one of the solutions proposed is to forcibly make all nodes unavailable [41]. But this trivial solution is unacceptable since it is unnecessarily tampers with remaining active nodes.

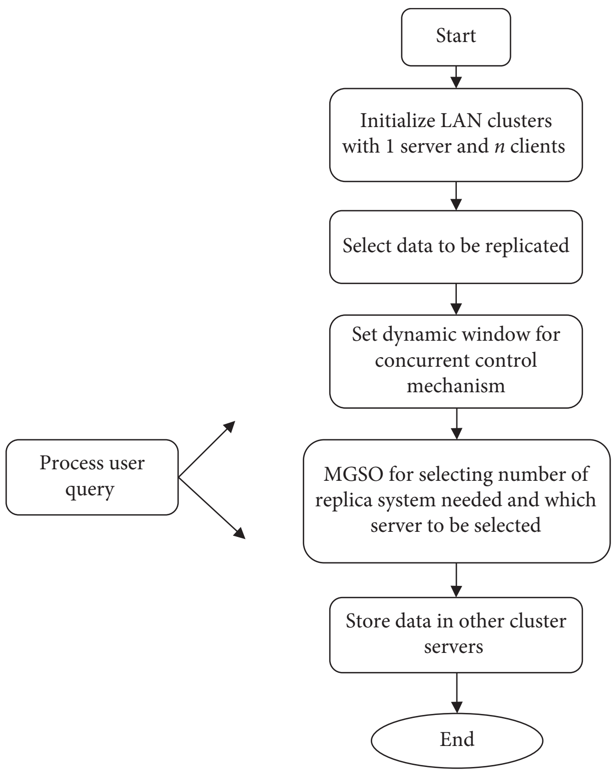
(iii) CAP theorem fails to encompass problems like node failure, loss, or delay of messages and restart time lapse of nodes other than partition. Fair link loss is possessed by a link if it has a nonzero probability of packet loss. In such a link, the lost packets will be delivered by a limited number of repeated attempts ensuring packets reaching the destination. The fair link loss is closely associated with mobile networks which are integral in today's application. Problems like node failures and restarts are no longer accidental as much as they occur due to attacks on a system. For instance, denial-of-service attack is common, and it is one of most notorious attacks on a network operation [42].

(iv) It is also possible to define consistency as a quantitative metric rather than a safety property. However, these stochastic definitions of consistency are not the subject of CAP.

(v) Partitioning is inevitable in any distributed system. Even if we assume that one node has 99.9% chance of not failing in a particular time period, five such nodes in a cluster will have a probability of 99.5% chance of failure [40]. Thus, one cannot compromise on partition tolerance. Therefore, there is an inevitable choice between availability and consistency.

4. MGSO-Based Dynamic Replication Strategy

The limitations of the CAP theorem have been considered, and this proposed model of the MGSO-based dynamic replication algorithm is presented in order to achieve better performance. Initially, the distributed database system to be considered is described. The scheme consists of n nodes, represented as $p_1, p_2, \ldots, p_n$. Each node comprises of a processor and a local memory. All the local memories are remote and available simply by local processors. Internode communication is supported by sending data through the underlying network. All requests, each with its corresponding time deadline, are presumed to reach the processors. Requests reach at a processor synchronously, and there is a concurrency control mechanism to serialize them for handling. For each requested data, it is anticipated that there are at least $t\,(1 \le t \le n)$ replicas in the database system, where n is the number of network nodes. This limitation is customarily referred to as t-availability constraint. Figure 1 shows the overall flow of the proposed dynamic replication strategy. The overall process begins with the initialization of the LAN clusters which consists of 1 server structure with n number of clients. The data to be replicated are selected, and they are loaded to the main server in the test environment. Then the dynamic window-based mechanism is utilized to access the data on each processor, where request of the same data is stocked on a temporal window until the fixed time deadline of the user request. This window mechanism determines the concurrent execution of the arriving user requests or queries. Based on this temporal window, the processor receiving more number of requests is considered for storing a replica of the original data from the main server in order to make it available for certain set of users. The MGSO is applied to identify the optimal number of processers that can be allowed to replicate based on queries/requests while also in determining which processors are suitable for storing the replica data depending upon their load and other features. The output obtained from the MGSO model is employed for final replica storage which becomes invalid if the publisher or authority of the original data tends to modify the data with updated contents. At these stages, the abovementioned process is restarted to update the replica system.


Start
Initialize LAN clusters with 1 server and n clients
Select data to be replicated
Set dynamic window for concurrent control mechanism
Process user query
MGSO for selecting number of replica system needed and which server to be selected
Store data in other cluster servers
End
FIGURE 1: MGSO-based dynamic data replication strategy.

In the proposed system, all requests reach the distributed processors in the network system. For evaluation purposes, we have utilized an example of storing replica of a dataset that contains YouTube, Sensor, and Twitter contents. These data are filtered and preprocessed to obtain complete individual set of data before storing them in the main server. Once the data are stored in the main server, the data requests from the users are analysed. When a processor p_i wants to read data d, if the latest version of data d is in its local memory, then data d are directly recovered from their local memory, else, since p_i knows the fixed processor set $S(d)$, p_i will direct a read request to the nearby server, p_j, in $S(d)$. This would sustain C_c units of cost. As a reply, p_j will recover data d from their local memory and send them to p_i, incurring $(C_{io} + C_d)$ units of cost. Finally, in order to minimize the total servicing cost of forthcoming requests, p_j might specify p_i to save data d into their local memory. This read request is described as the *saving-read* request. It should be noticed that exclusive servers of data d can get requests from remote processors to read data d. Furthermore, each processor can issue a write request for any data in a distributed database without loss of all inclusive statement. In this manner, the request arriving at processor q can be anyone of these: a read request from processor q for data d in their nearby memory and a write request from processor q for any data in the system, or a read request from a remote processor for a data d' if processor q is the server of data d'.

4.1. Cost Model. This cost model [43] is utilized to compute the cost of servicing a read request or a write request arriving at a processor q. The cost of servicing of request REQ by an algorithm A is defined as $\mathrm{COST}_A(\mathrm{Req})$. Consequently, in view of servicing a read request $R_d^{p_i}(k)$, let A_d be the allocation pattern of data d identified by processor q, and then the cost is given by

$$\mathrm{COST}_A\left(R_d^{p_i}(k)\right) = \begin{cases} 1 & \text{if } p_i \in A_d, \\ 1 + C_c + C_d & \text{if } p_i \notin A_d \text{ and } R_d^{p_i}(k) \text{ is not a saving} - \text{read request}, \\ 2 + C_c + C_d & \text{if } p_i \notin A_d \text{ and } R_d^{p_i}(k) \text{ is a saving} - \text{read request}. \end{cases} \tag{1}$$

An important feature of this model is that, after getting the data d, when p_i keeps data d into its local memory (saving-read request), then the servicing cost will be one unit greater than when p_i does not save data d (nonsaving-read request). Moreover, whether this read request is a saving-read request or not is certain by processor q established on the dynamic request window mechanism. Once processor q chooses that request $R_d^{p_i}(k)$ is a saving-read request, q and p_i will transform their corresponding allocation patterns.

Likewise, consider servicing a write request $W_d^q(k)$, and let A_d be the allocation pattern of data d recognized by processor q. Then, the cost of servicing this request is specified as follows:

$$\mathrm{COST}_A\left(W_d^q(k)\right) = \begin{cases} (t-1)C_d + t + \sum\limits_{p \in S(d)} |\mathrm{inv\_list}(p,d)| C_c & \text{if } q \in S(d), \\ tC_d + (t+1) + \sum\limits_{p \in S(d)} |\mathrm{inv\_list}(p,d) - \{q\}| C_c & \text{otherwise}, \end{cases} \tag{2}$$

where $\mathrm{inv\_list}(p,d) - \{q\}$ represents the set of processors in $\mathrm{inv\_list}(p,d)$, excluding processor q. A write request generates a different form of data. In order to sustain the consistency among the replicas of the data, the new form should be moved to those processors which have the available replicas of these data in their corresponding local memories. It must be noted that each relocation of these data to the corresponding processors will sustain C_d units of cost.

4.2. Window Mechanism. A certain concurrency control mechanism in each processor is expected to serialize the arriving requests so that it yields at most one request in a δ time units, with δ being imperceptibly small. Without loss of consensus, it is expected that $\delta = 1$. Each request landing at a processor creates an underlying request scheme. In order to process these requests, their individual time deadline is noted from the discharge points of the concurrency control mechanism and the window mechanism is invoked to respond to this request. Figure 2 shows the proposed window mechanism working process which defines the addition of new write requests to the currently available read requests. It can be seen that the concurrency control is performed in time slots for each requests.

Furthermore, the scheme of the window mechanism comprises a dynamic practice. Numerous dynamic request windows are produced in each processor, one for each requested object. A request window for data d in a processor is symbolized as $\mathrm{win}(d)$. Each request window is of a FIFO-type window with size τ to collect most τ number of requests in τ time units for the similar data. Furthermore, two counters, TC_d^1 and TC_d^2, are connected for each $\mathrm{win}(d)$. TC_d^1 has a preliminary value τ, and the value of TC_d^1 is decremented by one per time unit till it touches 0. TC_d^2 will track the time deadline of requests in $\mathrm{win}(d)$. The window mechanism is described in Algorithm 1.

It can be witnessed that the handling of requests in a request window can be determined by the deadline

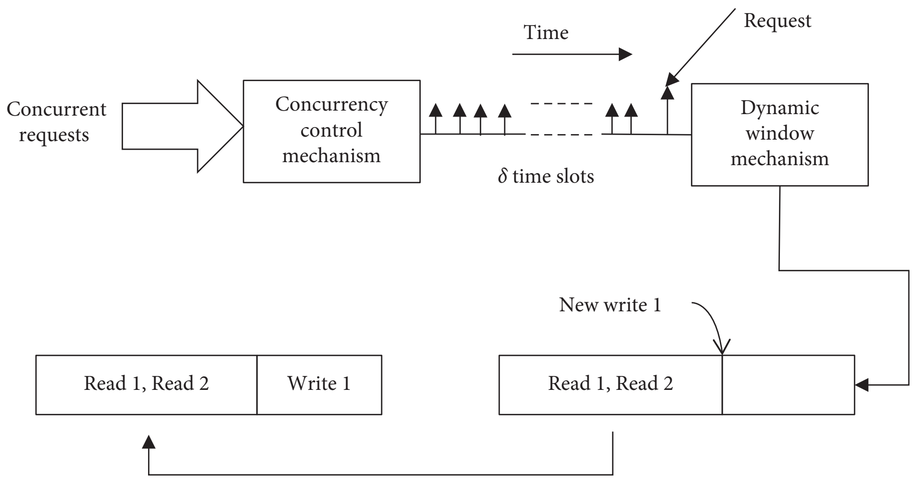


FIGURE 2: Working procedure of the dynamic window mechanism.

```
Initialize the servers and subservers
For each time unit
  If there is request REQ for data $d$ with time deadline $\tau_{\mathrm{REQ}}$
    If win$(d)$ does not exist
      Generate win$(d)$ and insert REQ into win$(d)$;
      $\mathrm{TC}_d^1=\tau$; $\mathrm{TC}_d^2=\tau_{\mathrm{REQ}}$;
    Else If REQ is a read request
      Insert REQ into win$(d)$; $\mathrm{TC}_d^2=\min(\mathrm{TC}_d^2,\tau_{\mathrm{REQ}})$;
    Else
      Service the requests in win$(d)$; Insert REQ into win$(d)$; $\mathrm{TC}_d^1=\tau$; $\mathrm{TC}_d^2=\tau_{\mathrm{REQ}}$;
    End if
  End if
End for
For each presently existing request window win$(d')$ in processor q
  $\mathrm{TC}_{d'}^1=\mathrm{TC}_{d'}^1-1;\mathrm{TC}_{d'}^2=\mathrm{TC}_{d'}^2-1$;
  If $(\mathrm{TC}_{d'}^1==0)$
    Service the requests in win$(d')$; Delete win$(d')$;
    If ($q$ is a server of data $d'$)
      Invalid the copies of $d'$ in processors that fit to inv_list$(q,d')$;
      Empty inv_list$(q,d')$;
    $A_{d'}=S(d')$
    End if
  Else if $(TC_{d'}^2==0)$
    Service the requests in win$(d')$; $TC_{d'}^1=\tau$; $TC_{d'}^2=\infty$;
  End if
End for
```

ALGORITHM 1: Window mechanism.

obligation of a specific request and is not utterly constrained by the realisation of the request window. In fact, three conditions can activate instant servicing of the requests in win(d). At any time, TC_d^1 of win(o) in processor q ranges 0, win(d) will be removed from q, and window mechanism will reset A_d in q to $S(d)$. Likewise, if q is a server of data d, then q will directly invalidate control-messages to the processors iinv_list(q,d)n inv_list(q,d) to invalidate the replicas of data d and void . The succeeding request for data d received at processor q will be deliberated as the first request for data d, and win(d) will be restarted by the window mechanism. Thus, the request structure introduced into win(d) is considered through its distinct lifetime as σ_d, and it is obvious that, by using the window mechanism, a σ_d will be principally subdivided into numerous phases $P(1)$, $P(2)$, . . ., $P(r)$. Each phase fits either into Type I or into Type II. While the Type I phase comprises of a number of read requests, a Type II phase comprises of a write request monitored by numerous read requests. Whenever a phase originates into a reality, the window mechanism will prove the request sequence in that phase without any understanding of the upcoming phases [44].

It must be noticed that the extent of a request window τ is a key parameter in our system. Different estimations of τ bring about various execution of the system. The estimation of τ ought to be resolved based on the node ability (for example, CPU power, memory limit, and network data transmission), system request arriving rate, and deadlines forced by the application requests. It is clear that the estimation of τ ought not to be too small. If $\tau = 1$, at that point, each request will frame a request sequence, and each request window would be produced and erased in one time unit. This will suddenly consume the vast majority of the computing capacity of the processor. It must also be noticed that if the deadlines imposed by the requests are too short for the system to process, certain requests might be dropped by the system. Without loss of generality, the deadline imposed by a request that can be effectively handled by the system is at least equivalent to 1. Such dropped requests known as the *blocked* requests would either leave the system or are resubmitted, depending on the underlying application. It can be managed by the admission control mechanism of this model.

4.3. Replica Selection and Placement Problem.

4.3. Replica Selection and Placement Problem. The selection of replica and the problem of placement in the distributed systems can be defined by considering a distributed system containing K data objects that would be replicated onto N servers. Let $S(n)$ and $O(k)$ be the names of server n and data object k, correspondingly. $C(n)$ and $V(k)$ denote the capacity of server n and the volume of data object k, respectively, where $1 \le n \le N$ and $1 \le k \le K$. A link amongst two servers $S(n)$ and $S(m)$ (if it occurs) has an integer number $l(n,m)$, and this provides the communication cost for transmitting a data unit amongst servers $S(n)$ and $S(m)$. It is presumed that $l(n,m) = l(m,n)$. Let $\text{read}(n,k)$ and $\text{write}(n,k)$ be the number of read and write demands requested from server n for data object k. Each data object $O(k)$ has a principal server $P(k)$, which holds the main copy of $O(k)$. The main copy of data objects cannot be deallocated. Figure 3 shows the replica placement strategy in which the replica has been selected after determining the constraints and then the placement is carried out for practical use.

It is expected that each primary server has the replication representation of k-th data object, $\text{RS}(k)$, which comprises a list of servers where $\text{RS}(k)$ is replicated. In order to achieve a write request, $P(k)$ obtains the update request from the source server which requests to update the data object $O(k)$ and broadcasts it to all servers in its replication representation $\text{RS}(k)$. The main objective of replica placement problem is allocating replicas over all the servers in order to reduce the total operation cost $\text{TOC}(k)$, which is induced by two modules,$\text{access}_\text{R}(k)$ and $\text{access}_\text{w}(k)$. Hence, $\text{TOC}(k)$ is obtained as follows:

$$\text{TOC}(k) = \text{access}_\text{R}(k) + \text{access}_\text{w}(k), \tag{3}$$

where $\text{access}_\text{R}(k)$ is the total operation cost due to all servers reading requests for $O(k)$ and $\text{access}_\text{w}(k)$ the total operation cost due to all servers writing requests for $O(k)$.

$\text{access}_\text{R}(k)$ and $\text{access}_\text{w}(k)$ are given as follows:

$$\begin{aligned}\text{access}_\text{R}(k) &= V(k) \times \left(\sum_{n=1}^{N} \text{read}(n,k) \times l(n,..,\text{NS}(n,k)) \right),\\ \text{access}_\text{w}(k) &= V(k) \times \sum_{n=1}^{N} \Bigg(\text{write}(n,k)\\ &\quad \times \Bigg[l(n,P(k)) + \sum_{(\forall j \in \text{RS}(k)), j \ne n} l(P(k),j) \Bigg] \Bigg),\end{aligned} \tag{4}$$

where $\text{NS}(n,k)$ is the nearby server to $S(n)$ that encloses a replica of $O(k)$.

Consequently, the concise TOC based on total read and write requests' cost for all data objects is obtained as follows:

$$\text{TOC} = \sum_{k=1}^{K} \text{TOC}(k). \tag{5}$$

Reducing this equation is the solution to the replica selection and placement problem in distributed systems. This is achieved in this proposed model by using a multi-objective glowworm swarm optimization algorithm for optimally selecting the number of replica and also with the server location for placement of replica.

4.4. MGSO-Based Replica Selection and Placement. In this proposed model, each glow-worm agent (gwa) is a $N \times K$ matrix with Boolean components [45]. The replication problem is modelled into each glow-worm, and it moves towards the brighter glowworm, i.e., determining which processor is better suited to store the replica data. As the utilized glowworm and agents are of $N \times K$ Boolean matrix, the logical operations are employed. The logical OR operations are performed by the inbuilt processing unit to move the selected glowworm towards the better one in terms of brightness (i.e., better replica conditions). Let element $\text{gwa}_{n,k} = 1$ if $S(n)$ comprises a replica of $O(k)$ and $\text{gwa}_{n,k} = 0$ otherwise. The fitness value of each agent is calculated in terms of TOC percentage, which is saved using the replication strategy of the algorithms, compared to the initial one, i.e., when only primary copies exist. This value indicates the solution quality of replication schema associated with the glowworm agent. Fitness function is computed as follows:

$$f_{\text{gwa}(i)} = 1 - \frac{\text{TOC}_{\text{gwa}(i)}}{\text{TOC}_{\text{initial}}}, \tag{6}$$

where $\text{TOC}_{\text{gwa}(i)}$ is the TOC for replication scheme for the i-th agent and $\text{TOC}_{\text{initial}}$, the TOC achieved in the initial allocation, which is calculated only when the primary copy of the data objects exists while replica has not been produced.

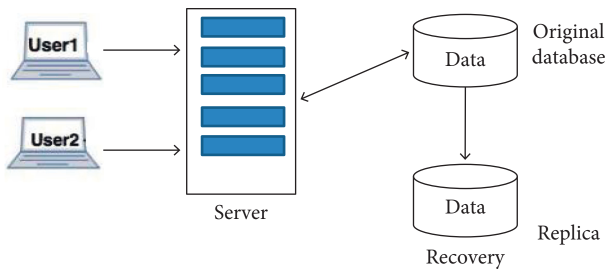


Figure 3: Replica placement strategy.

In addition to the Boolean logical operations, the update and other processes of the MGSO are also needed to be processed which are undertaken by the agents specified. The validity of each gwa is influenced by the servers' capacity. The foremost principles in a multiobjective GSO is that the solution is not an absolute optimal solution as in the common GSO because only the most viable among the optimal solution is chosen, and this cannot be consider as 100% optimal. The population and step size are initialized randomly for the noninferior sorting process. The luciferin update process is performed as follows:

$$l_i(t) = (1-\rho)l_i(t-1) + \gamma J(x_i(t)), \tag{7}$$

where $l_i(t)$ represents the luciferin level associated with glowworm i at time t, ρ is the luciferin decay constant $(0<\rho<1)$, γ is the luciferin enhancement constant, and $J(x_i(t))$ represents the value of the objective function at glowworm i-th location at time t.

During the movement phase, each glowworm decides, using a probabilistic mechanism, to move toward a neighbor that has a luciferin value higher than its own. However, in order to increase population diversity and the convergence speed, a disturbance term is added to the location updating formula:

$$X_i(t+1) = X_i(t) + s\left(\frac{X_j(t)-X_i(t)}{\left\|X_j(t)-X_i(t)\right\|}\right) + u, \tag{8}$$

where $u = \alpha * \text{rand} * (\text{iter\_max} - t)/\text{iter\_max}$, s step, and α which can control the range of disturbance, rand, the random number that meets normal distribution. α is the algorithm parameter set in advance, which is relevant to the problem. The range of α values 0 to 1, but in general, the value of α is set to 0.001.

When the glowworms are determined only by the local information to select their movements, it is anticipated that the number of peaks caught would be a function of the radial sensor range. Consequently, GSO utilizes an adaptive neighbourhood range in order to detect the existence of multiple peaks in a multimodal utility landscape. It is represented as follows:

$$r_d^i(t+1) = \min\left\{r_s, \max\left\{0, r_d^i(t) + \beta\left(x_t - \left|X_i(t)\right|\right)\right\}\right\}, \tag{9}$$

where r_d^i is the neighbourhood range, r_s is the neighbourhood range with respect to step size, β is a constant parameter, and x_t is a parameter used to control the number of neighbors. The complete process of MGSO for replication process in distributed systems is given as shown in Algorithm 2.

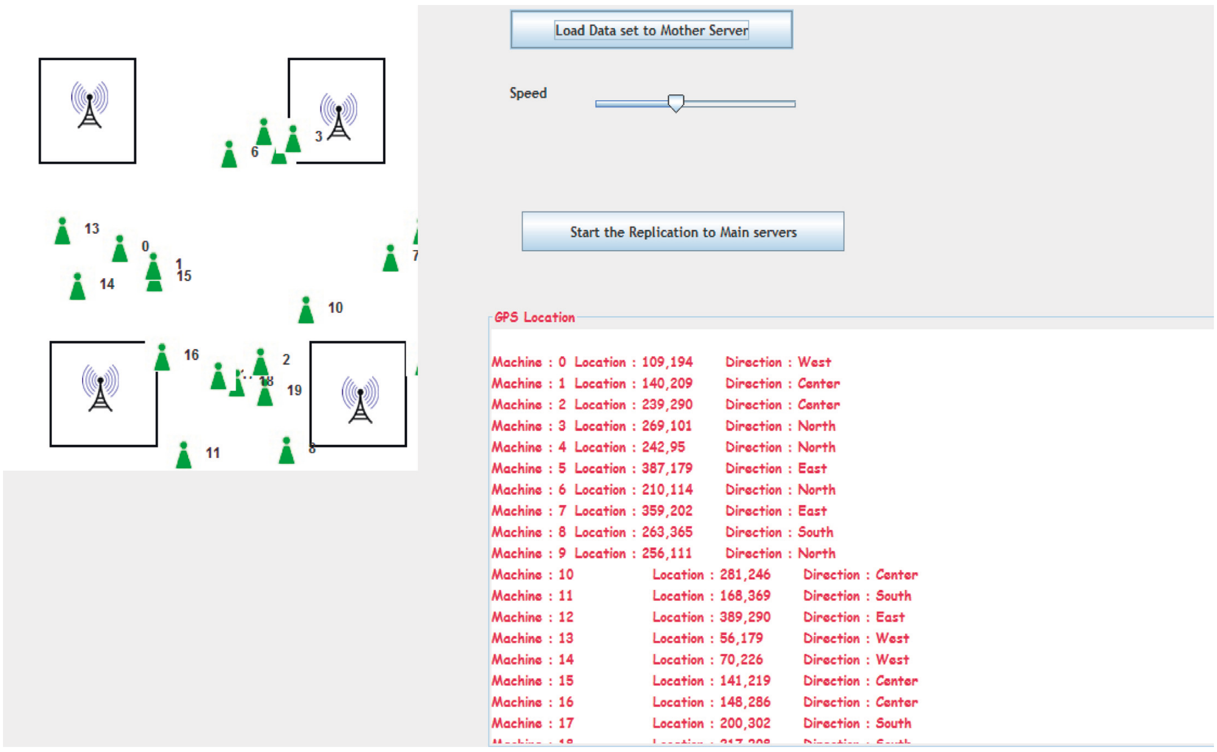
```
Initialize the population $N$, step size $s$
Set servers (solutions) as gwa
$A = 0$; All dominated solution will be placed in $A$
Let $t = 0$;
Compute $f_{\text{gwa}(i)}$;
Update $l_i(t)$;
Determine direction of movement of each gwa;
Update gwa location;
Noninferior sorting process initiated;
Update $A$;
$t = t + 1$;
Until $(t = \text{iter\_max})$
Place replica at $S(n)$ in $A$;
End
```

Algorithm 2: MGSO-based replica placement.

The most time-consuming parts of the algorithms are associated with population initialization, and updating best positions is achieved by all glowworms in each iteration. The time complexity of initializing N *glowworms* is $O(N_p(K^2N + KN^2))$. In order to update best neighbourhood locations in each iteration, fitness values of all *gwa* must be considered. The time complexity of fitness value calculation is $O(K^2N)$. Thus, the time complexity of solving replica placement in distributed systems via MGSO is $O(N_p(K^2N + KN^2) + N_g(K^2N))$.

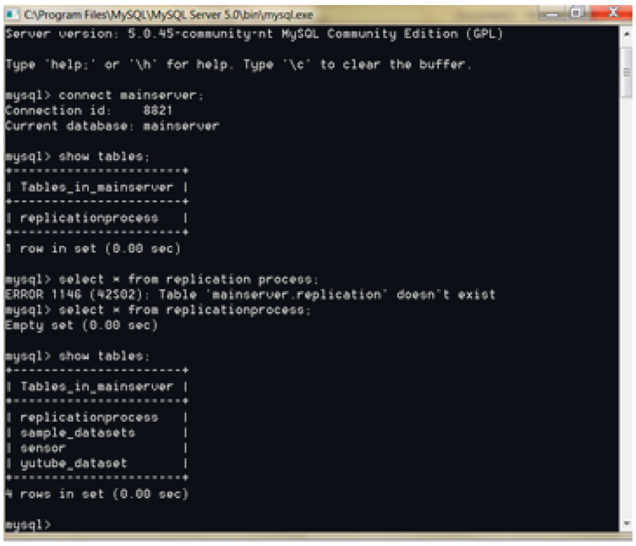
5. Simulation Results

5.1. Validation of MGSO-Based Replication Algorithm in Hadoop Simulation. The proposed MGSO-based dynamic replication algorithm uses the strategies of MGSO with the dynamic window mechanism to tackle the limitations of the CAP theorem. The theoretical soundness of any algorithm becomes legated when it is practically not viable. Hence, the validation of the MGSO-based algorithm is performed in Hadoop clusters. The Hadoop framework consists of Hadoop distributed file system (HDFS) and MapReduce framework, which seem efficient in setting up multiple clusters of LAN servers and subservers for evaluation. The HDFS enables to utilize it for memory purpose, without requiring the user to purchase cloud or other storage platforms. The cluster set up ensures that there is adequate data transmission between the client systems in distant location without much packet loss and also with


Load Data set to Mother Server
Speed
Start the Replication to Main servers
GPS Location

```
Machine : 0  Location : 109,194      Direction : West
Machine : 1  Location : 140,209      Direction : Center
Machine : 2  Location : 239,290      Direction : Center
Machine : 3  Location : 269,101      Direction : North
Machine : 4  Location : 242,95       Direction : North
Machine : 5  Location : 387,179      Direction : East
Machine : 6  Location : 210,114      Direction : North
Machine : 7  Location : 359,202      Direction : East
Machine : 8  Location : 263,365      Direction : South
Machine : 9  Location : 256,111      Direction : North
Machine : 10            Location : 281,246      Direction : Center
Machine : 11            Location : 168,369      Direction : South
Machine : 12            Location : 389,290      Direction : East
Machine : 13            Location : 56,179       Direction : West
Machine : 14            Location : 70,226       Direction : West
Machine : 15            Location : 141,219      Direction : Center
Machine : 16            Location : 148,286      Direction : Center
Machine : 17            Location : 200,302      Direction : South
```

(a)



```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
Server version: 5.0.45-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> connect mainserver;
Connection id:    8821
Current database: mainserver

mysql> show tables;
+----------------------+
| Tables_in_mainserver |
+----------------------+
| replicationprocess   |
+----------------------+
1 row in set (0.00 sec)

mysql> select * from replication process;
ERROR 1146 (42S02): Table 'mainserver.replication' doesn't exist
mysql> select * from replicationprocess;
Empty set (0.00 sec)

mysql> show tables;
+----------------------+
| Tables_in_mainserver |
+----------------------+
| replicationprocess   |
| sample_datasets      |
| sensor               |
| yutube_dataset       |
+----------------------+
4 rows in set (0.00 sec)

mysql>
```

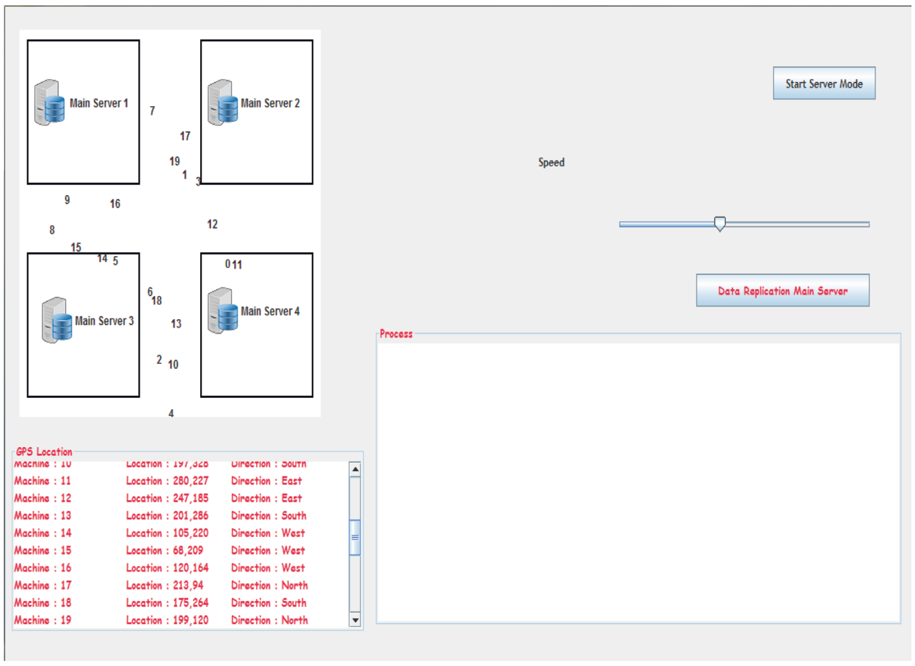
(b)

Figure 4: (a) Initial server set up for primary data copy. (b) Primary data storage process.

minimal energy consumption. As said above, for validation, a raw database consisting of data which is collected from three different domains is used. The raw database includes environmental data collected by sensors and social data collected from Twitter and also YouTube. Initially, these three types of data are unstructured, which will eventually be refined during the replication primary copy storage time using the window mechanism. Let the validation results help us in justifying the performance of the proposed model.

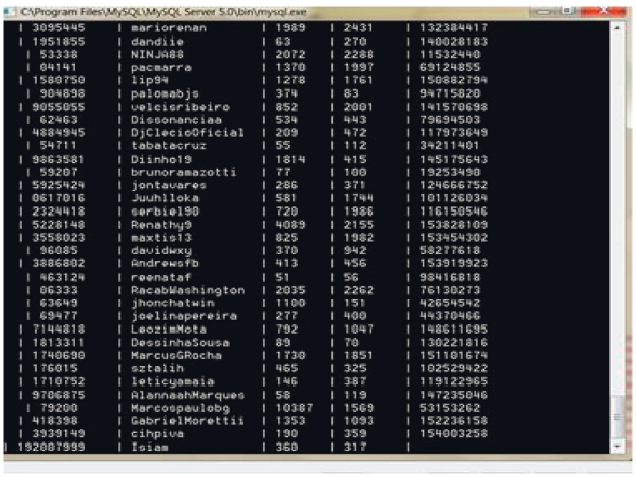
Let us take the abovementioned database for simulating the proposed algorithm. The initial Hadoop settings are done as in the instructions manual, and then the main project for simulation is done. Figure 4 shows the initial processes involved in setting up primary data copy in a distributed system for replication. These data are the main source that contain all the three types of collected data, namely YouTube, Sensor, and Twitter.

Figure 5 shows the effect of window mechanism for the replication process. Figure 5(a) illustrates the practical



```
Machine : 10    Location : 197,328    Direction : South
Machine : 11    Location : 280,227    Direction : East
Machine : 12    Location : 247,185    Direction : East
Machine : 13    Location : 201,286    Direction : South
Machine : 14    Location : 105,220    Direction : West
Machine : 15    Location : 68,209     Direction : West
Machine : 16    Location : 120,164    Direction : West
Machine : 17    Location : 213,94     Direction : North
Machine : 18    Location : 175,264    Direction : South
Machine : 19    Location : 199,120    Direction : North
```

(a)



```
C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql.exe
 | 3095445    | mariorenan       | 1989  | 2431 | 132384417
 | 1951855    | dandiie          | 63    | 270  | 140028183
  | 53338     | NINJA88          | 2072  | 2288 | 11532440
  | 84141     | pacmarra         | 1370  | 1997 | 69124855
 | 1580750    | lip94            | 1278  | 1761 | 150882794
  | 304898    | palomabjs        | 374   | 83   | 94715820
 | 9055055    | velcisribeiro    | 852   | 2001 | 141570698
  | 62463     | Dissonanciaa     | 534   | 443  | 79694503
 | 4884945    | DjClecioOficial  | 209   | 472  | 117973649
  | 54711     | tabatacruz       | 55    | 112  | 34211401
 | 9863581    | Diinho19         | 1814  | 415  | 145175643
  | 59207     | brunoramazotti   | 77    | 100  | 19253490
 | 5925424    | jontavares       | 286   | 371  | 124666752
 | 0617016    | Juuhlloka        | 581   | 1744 | 101126034
 | 2324418    | serbiel90        | 720   | 1986 | 116150546
 | 5228148    | Renathy9         | 4089  | 2155 | 153828109
 | 3558023    | maxtis13         | 825   | 1982 | 153454302
  | 96085     | davidwxy         | 370   | 942  | 58277618
 | 3886802    | Andrewsfb        | 413   | 456  | 153919923
  | 463124    | reenataf         | 51    | 56   | 98416818
  | 86333     | RacabWashington  | 2035  | 2262 | 76130273
  | 63649     | jhonchatwin      | 1100  | 151  | 42654542
  | 69477     | joelinapereira   | 277   | 400  | 44370466
 | 7144818    | LeozimMota       | 792   | 1047 | 148611695
 | 1813311    | DessinhaSousa    | 89    | 70   | 130221816
 | 1740690    | MarcusGRocha     | 1730  | 1851 | 151101674
 | 176015     | sztalih          | 465   | 325  | 102529422
 | 1710752    | leticyamaia      | 146   | 387  | 119122965
 | 9706875    | AlannaahMarques  | 58    | 119  | 147235046
  | 79200     | Marcospaulobg    | 10387 | 1569 | 53153262
 | 418398     | GabrielMorettii  | 1353  | 1093 | 152236158
 | 3939149    | cihpiva          | 190   | 359  | 154003258
| 192087999   | Isiam            | 360   | 317  |
```

(b)

Figure 5: (a) Window initialization before replication. (b) Data categorization using window mechanism.

settings required for initializing the window mechanism, while Figure 5(b) shows the query/request analysis based on the data categorization of those requests. Based on the requests, the demand for each datum is obtained which is helpful in determining the number of replica.

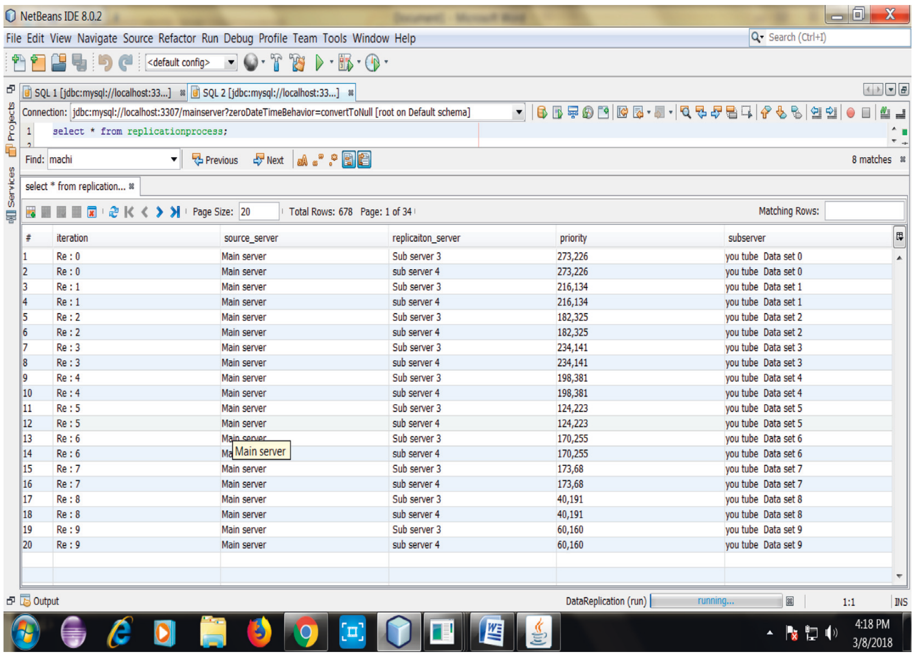
Figure 6 shows the backend server table after the admission control mechanism where the storage and processing capabilities of each server/processor are stored. This information will be utilized for selecting the processors where the data have to be replicated. Figure 7 shows the replica selection and placement process as performed based on the concept of Figure 3 utilizing the MGSO algorithm.

Figure 8 shows the replica selection using MGSO. This process is achieved after the initialization of the window mechanism. When the raw data are categorized into three individual categories, the subserver which is most suitable for storing specific category is chosen using MGSO.

Figure 9 shows the MGSO-based replica placement stage. In this stage, the replica which is basically a single category of data are placed in the relevant server in another cluster. These data can be fetched through a user query by the clients as and when required.

Figure 10 shows the data replication placement table which is maintained at the servers in each cluster as a routing table for priority selection when any user query arises. Based on these tables, the servers in the clusters select the best server to fetch the data relevant to the query when a client raises a query.

The query process is described in Figures 11(a) and 11(b). When a user requires query, they intimate the query to the client system, which in turn is forwarded to the


NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
SQL 1 [jdbc:mysql://localhost:33...]
SQL 2 [jdbc:mysql://localhost:33...]
Projects
Services
Find: machi
Previous
Next
8 matches
select * from replication...
Matching Rows:
Output
DataReplication (run)
running...
1:1
INS
4:18 PM
3/8/2018

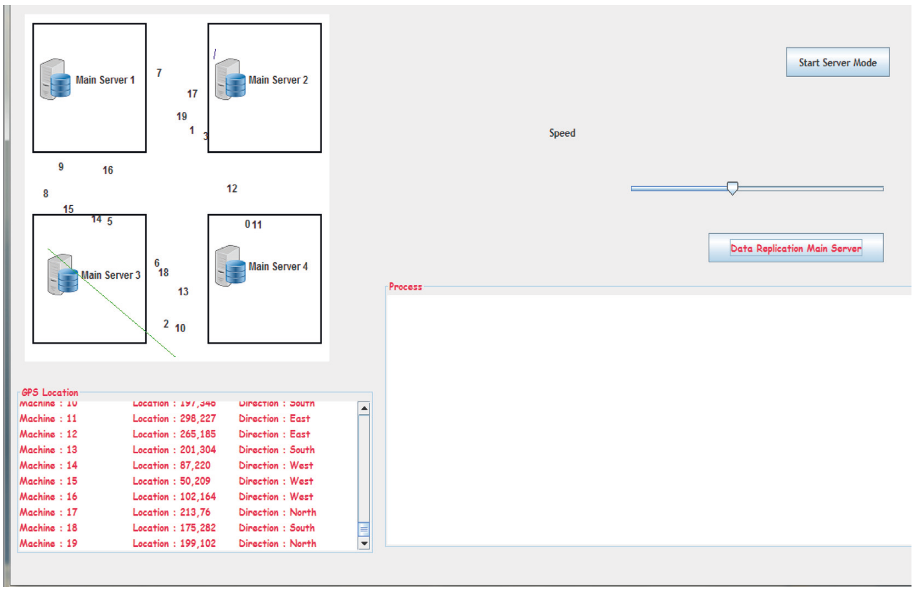
Connection: jdbc:mysql://localhost:3307/mainserver?zeroDateTimeBehavior=convertToNull [root on Default schema]

```
select * from replicationprocess;
```

Page Size: 20 | Total Rows: 678 Page: 1 of 34

#	iteration	source_server	replicaiton_server	priority	subserver
1	Re : 0	Main server	Sub server 3	273,226	you tube Data set 0
2	Re : 0	Main server	sub server 4	273,226	you tube Data set 0
3	Re : 1	Main server	Sub server 3	216,134	you tube Data set 1
4	Re : 1	Main server	sub server 4	216,134	you tube Data set 1
5	Re : 2	Main server	Sub server 3	182,325	you tube Data set 2
6	Re : 2	Main server	sub server 4	182,325	you tube Data set 2
7	Re : 3	Main server	Sub server 3	234,141	you tube Data set 3
8	Re : 3	Main server	sub server 4	234,141	you tube Data set 3
9	Re : 4	Main server	Sub server 3	198,381	you tube Data set 4
10	Re : 4	Main server	sub server 4	198,381	you tube Data set 4
11	Re : 5	Main server	Sub server 3	124,223	you tube Data set 5
12	Re : 5	Main server	sub server 4	124,223	you tube Data set 5
13	Re : 6	Main server	Sub server 3	170,255	you tube Data set 6
14	Re : 6	Main server	sub server 4	170,255	you tube Data set 6
15	Re : 7	Main server	Sub server 3	173,68	you tube Data set 7
16	Re : 7	Main server	sub server 4	173,68	you tube Data set 7
17	Re : 8	Main server	Sub server 3	40,191	you tube Data set 8
18	Re : 8	Main server	sub server 4	40,191	you tube Data set 8
19	Re : 9	Main server	Sub server 3	60,160	you tube Data set 9
20	Re : 9	Main server	sub server 4	60,160	you tube Data set 9

Figure 6: Backend server memory table.

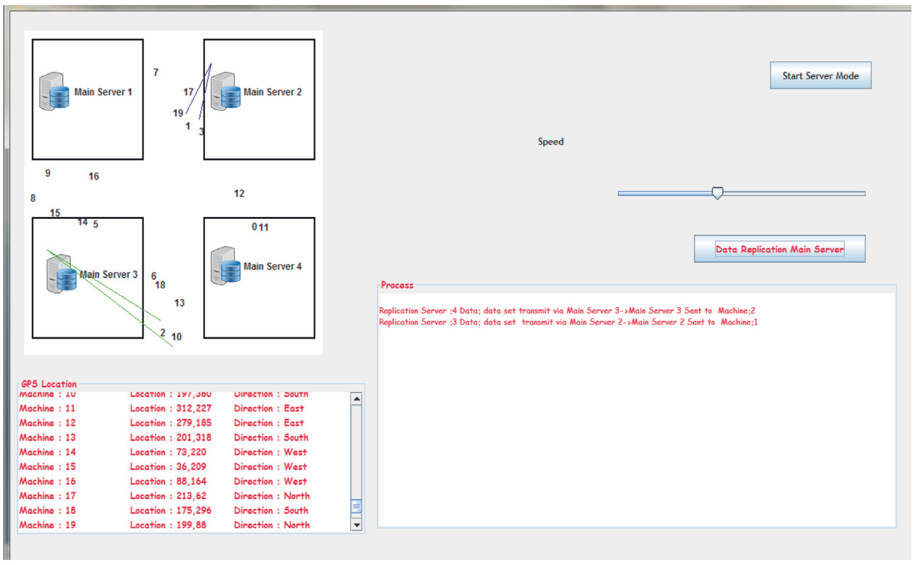


```
Machine : 10        Location : 197,346    Direction : South
Machine : 11        Location : 298,227    Direction : East
Machine : 12        Location : 265,185    Direction : East
Machine : 13        Location : 201,304    Direction : South
Machine : 14        Location : 87,220     Direction : West
Machine : 15        Location : 50,209     Direction : West
Machine : 16        Location : 102,164    Direction : West
Machine : 17        Location : 213,76     Direction : North
Machine : 18        Location : 175,282    Direction : South
Machine : 19        Location : 199,102    Direction : North
```

Figure 7: Replica selection and placement process.

respective server. As mentioned before, the server checks the availability of the data for the query, and based on the priority of stored server locations, it chooses the server with minimum time consumption time. The request REQ is then forwarded, and the data are obtained for the write requests, as shown in Figure 11(b).

5.2. Performance Evaluation. The simulations are done using the Hadoop system, and the performance of the proposed MGSO-based model is compared with that of other algorithms. The performance of the MGSO-based replication algorithm is compared with the following four replication algorithms, namely, least frequently used (LFU), least recently used (LRU), 3-level hierarchical algorithm (3LHA), and bandwidth hierarchy-based replication algorithm (BHR). Execution time is how long it takes a program to run while the time complexity is the asymptotic behavior of running time as input size tends to infinity. In this evaluation, mean execution time is used which is the average of time taken by the algorithm to select the optimal replica and subsequent placement and the time to execute one read/write request for 12 iterations. We have utilized this parameter to evaluate the efficiency of the proposed model.



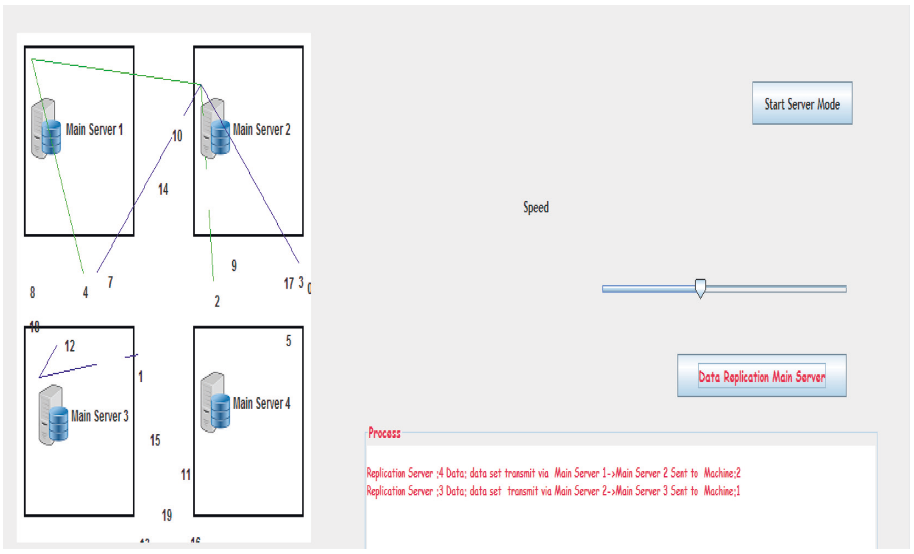
GPS Location

```
Machine : 10        Location : 197,360     Direction : South
Machine : 11        Location : 312,227     Direction : East
Machine : 12        Location : 279,185     Direction : East
Machine : 13        Location : 201,318     Direction : South
Machine : 14        Location : 73,220      Direction : West
Machine : 15        Location : 36,209      Direction : West
Machine : 16        Location : 88,164      Direction : West
Machine : 17        Location : 213,62      Direction : North
Machine : 18        Location : 175,296     Direction : South
Machine : 19        Location : 199,88      Direction : North
```

Process

```
Replication Server ;4 Data; data set transmit via Main Server 3->Main Server 3 Sent to  Machine;2
Replication Server ;3 Data; data set  transmit via Main Server 2->Main Server 2 Sent to  Machine;1
```

Figure 8: Replica selection using MGSO.



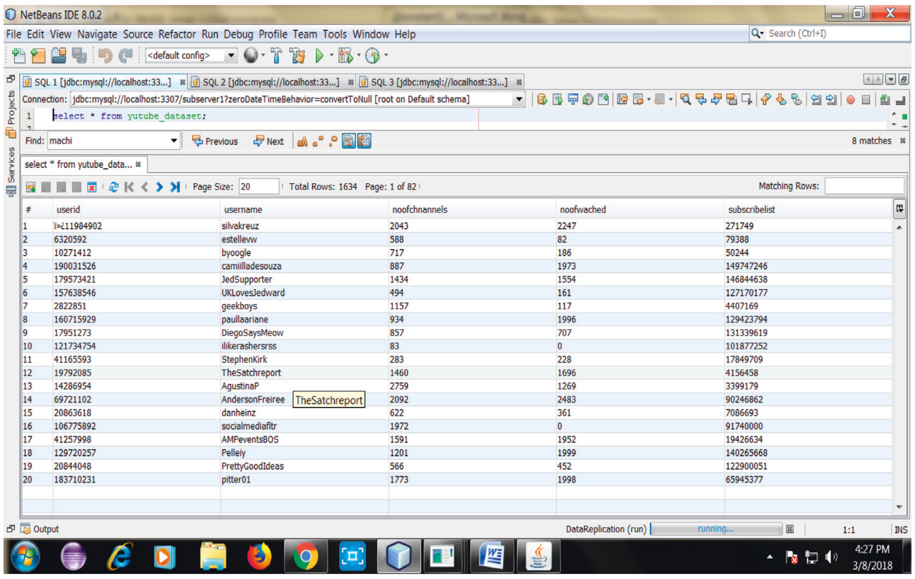
Process

```
Replication Server ;4 Data; data set transmit via  Main Server 1->Main Server 2 Sent to  Machine;2
Replication Server ;3 Data; data set  transmit via Main Server 2->Main Server 3 Sent to  Machine;1
```

Figure 9: MGSO-based replica placement.

Figure 12 illustrates the mean execution time comparison of the replication algorithms with respect to the storage size, while Figure 13 shows the comparison of the algorithms in terms of mean execution time with respect to file size. It can be observed that the MGSO-based dynamic replication algorithm has lesser execution time when compared to the other algorithms. Similarly, even when larger files are employed for replication, the proposed model has less execution time, thus justifying the performance of the model. The reason that can validate this deviation is the fact that the window mechanism and optimal replica selection concept significantly reduce the execution time.

Figure 14 shows the execution time for replica selection and placement, and Figure 15 shows the execution time of 1 read request for different file sizes. The execution time of the other algorithms are considerably higher than the proposed MGSO-based model as the optimal number of replica selection is performed within less time and the subsequent selection of processors is also optimal. Due to this reason, the MGSO algorithm has better performance in the selection process. Similarly, the proposed model consumes less time in executing 1 read request with different file sizes. Even with increased file size, the proposed model outperforms the other models with faster execution. It also hosts an auxiliary processor that has programmable accelerators to perform various tasks more efficiently and is attached to a number of peripherals enabling efficient performance. In this paper, the mean execution time is considered as the main efficiency criteria for analysing data reusability and parallelization. In this regard, the alternative configurations are comparable. In order to determine the best implementation, several options were explored. The simplest is to perform the entire


NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
SQL 1 [jdbc:mysql://localhost:33...]
SQL 2 [jdbc:mysql://localhost:33...]
SQL 3 [jdbc:mysql://localhost:33...]
Projects
Services
Find: machi
Previous
Next
8 matches
Output
DataReplication (run)
running...
1:1
INS
4:27 PM
3/8/2018

Connection: jdbc:mysql://localhost:3307/subserver1?zeroDateTimeBehavior=convertToNull [root on Default schema]

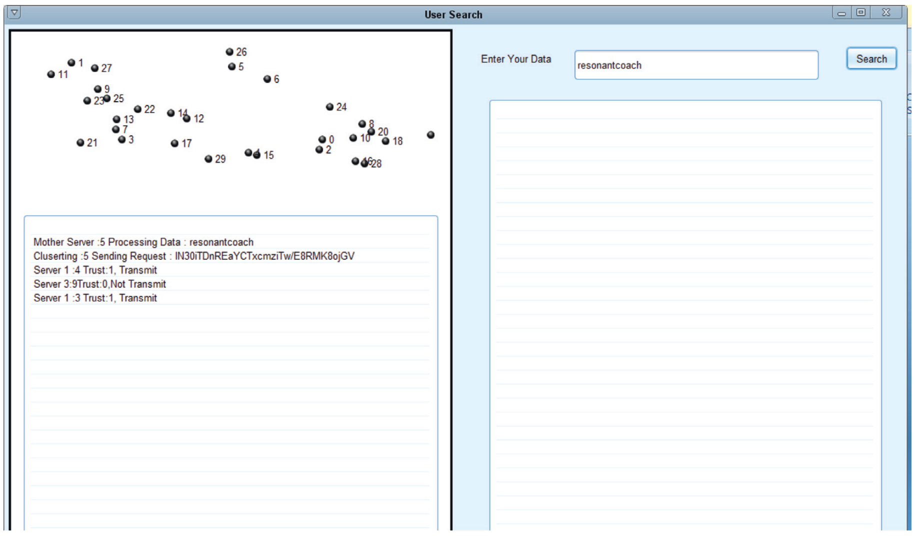
```
select * from yutube_dataset;
```

select * from yutube_data...

Page Size: 20 | Total Rows: 1634 Page: 1 of 82 | Matching Rows:

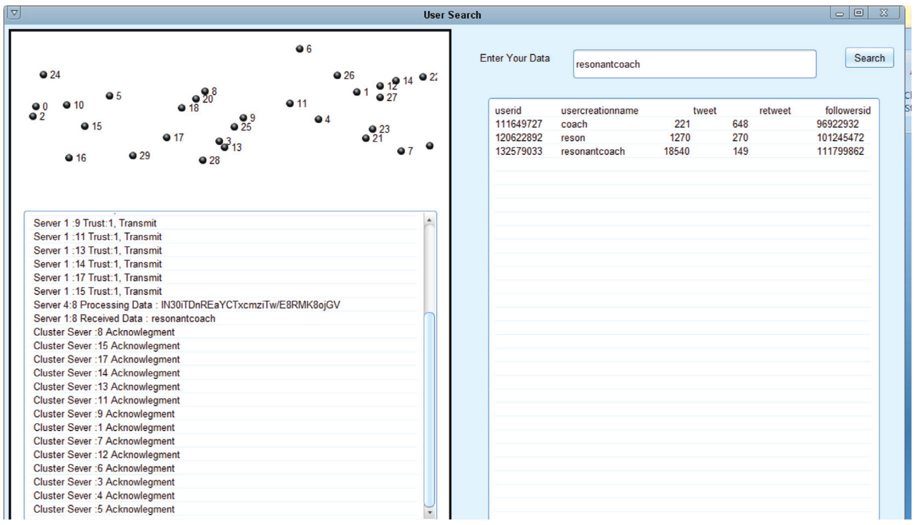
#	userid	username	noofchnannels	noofwached	subscribelist
1	ï»¿11984902	silvakreuz	2043	2247	271749
2	6320592	estellevw	588	82	79388
3	10271412	byoogle	717	186	50244
4	190031526	camiilladesouza	887	1973	149747246
5	179573421	JedSupporter	1434	1554	146844638
6	157638546	UKLovesJedward	494	161	127170177
7	2822851	geekboys	1157	117	4407169
8	160715929	paullaariane	934	1996	129423794
9	17951273	DiegoSaysMeow	857	707	131339619
10	121734754	ilikerashersrss	83	0	101877252
11	41165593	StephenKirk	283	228	17849709
12	19792085	TheSatchreport	1460	1696	4156458
13	14286954	AgustinaP	2759	1269	3399179
14	69721102	AndersonFreiree TheSatchreport	2092	2483	90246862
15	20863618	danheinz	622	361	7086693
16	106775892	socialmediafltr	1972	0	91740000
17	41257998	AMPeventsBOS	1591	1952	19426634
18	129720257	Pelleiy	1201	1999	140265668
19	20844048	PrettyGoodIdeas	566	452	122900051
20	183710231	pitter01	1773	1998	65945377

Figure 10: Data replication placement table.



```
Mother Server :5 Processing Data : resonantcoach
Cluserting :5 Sending Request : IN30iTDnREaYCTxcmziTw/E8RMK8ojGV
Server 1 :4 Trust:1, Transmit
Server 3:9Trust:0,Not Transmit
Server 1 :3 Trust:1, Transmit
```

(a)



```
Server 1 :9 Trust:1, Transmit
Server 1 :11 Trust:1, Transmit
Server 1 :13 Trust:1, Transmit
Server 1 :14 Trust:1, Transmit
Server 1 :17 Trust:1, Transmit
Server 1 :15 Trust:1, Transmit
Server 4:8 Processing Data : IN30iTDnREaYCTxcmziTw/E8RMK8ojGV
Server 1:8 Received Data : resonantcoach
Cluster Sever :8 Acknowlegment
Cluster Sever :15 Acknowlegment
Cluster Sever :17 Acknowlegment
Cluster Sever :14 Acknowlegment
Cluster Sever :13 Acknowlegment
Cluster Sever :11 Acknowlegment
Cluster Sever :9 Acknowlegment
Cluster Sever :1 Acknowlegment
Cluster Sever :7 Acknowlegment
Cluster Sever :12 Acknowlegment
Cluster Sever :6 Acknowlegment
Cluster Sever :3 Acknowlegment
Cluster Sever :4 Acknowlegment
Cluster Sever :5 Acknowlegment
```

userid	usercreationname	tweet	retweet	followersid
111649727	coach	221	648	96922932
120622892	reson	1270	270	101245472
132579033	resonantcoach	18540	149	111799862

(b)

Figure 11: (a) User query process: input query. (b) User query process: final result.

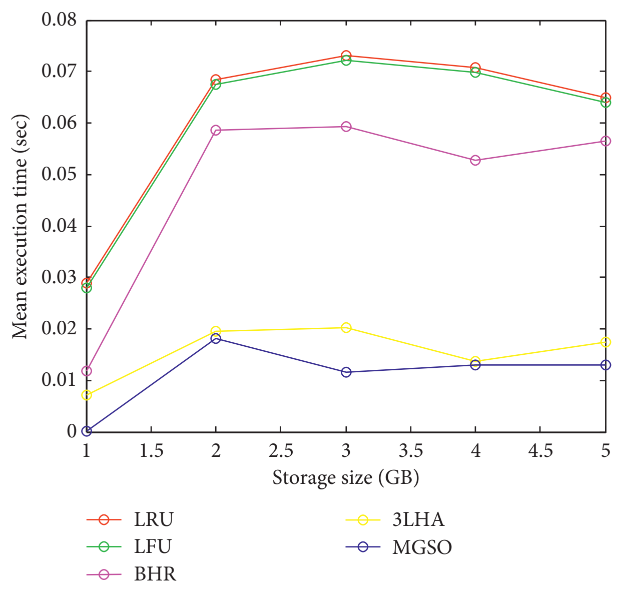


Figure 12: Storage size vs. mean execution time.

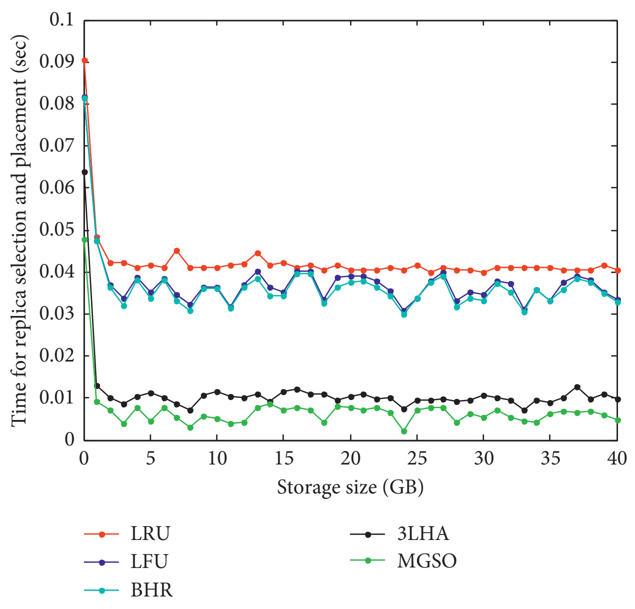


Figure 14: Execution time of replica selection and placement.

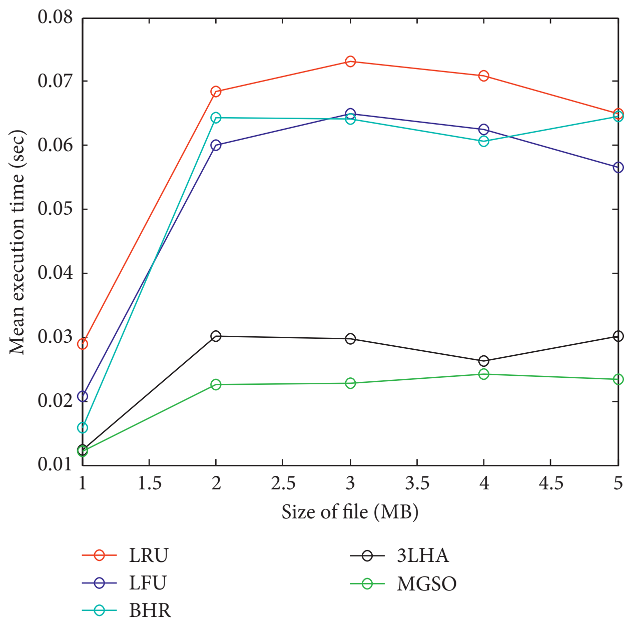


Figure 13: Size of file vs. mean execution time.

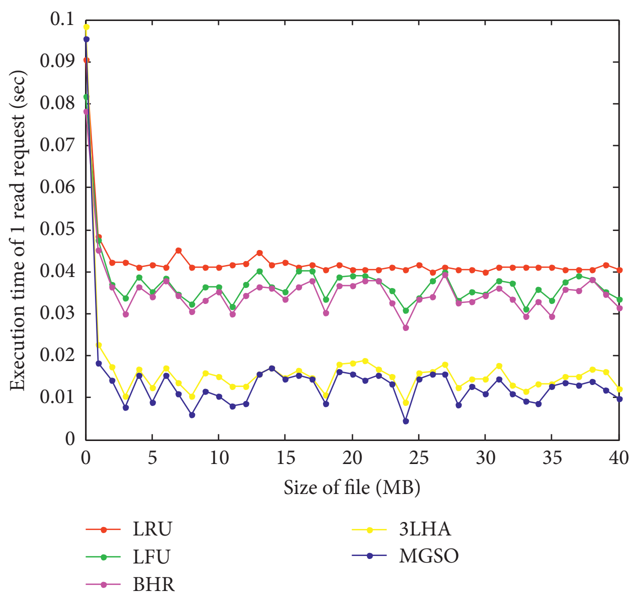


Figure 15: Execution time of 1 read request for different file sizes.

algorithm on the host processor; this approach was used as a benchmark to assess the improvements made by the different implementations. Even though this approach incurs some negligible overhead due to replica operations, the improvement in execution time favours this approach. Thus, from the performance evaluation, it can be concluded that the proposed MGSO-based dynamic replication algorithm is the better algorithm.

6. Conclusion

Distributed systems are the highlights of most data-intensive computing application. These systems predominate most web services and geographic data processing systems. The issues of selecting optimal replica number and replica server location still continue to throw challenges that prove to be roadblocks to efficient performance the distributed systems' applications. This study focused on developing a dynamic data replication algorithm for real-time distributed databases using MGSO that resolves the existing open challenges to a considerable extend. The simulation results also illustrate the efficient performance of the proposed MGSO-based replication algorithm. In future, MGSO-based replication algorithm can be combined with a proper scheduling algorithm to improve the overall performance of the system. The model can also be extended with additional parameters such as fault tolerance and security.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*, Prentice-Hall, Upper Saddle River, NJ, USA, 2007.

[2] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, Springer Science and Business Media, Berlin, Germany, 2011.

[3] S. W. Ambler, *Mapping Objects to Relational Databases: What You Need to Know and Why*, Ronin International, London, England, 2000.

[4] J. W. Yoder, R. E. Johnson, and Q. D. Wilson, *Connecting Business Objects to Relational Databases,* Vol. 51, University of Illinois, Urbana, Kolkata, 2005.

[5] P. Padmanabhan, L. Gruenwald, A. Vallur, and M. Atiquzzaman, "A survey of data replication techniques for mobile ad hoc network databases," *VLDB Journal*, vol. 17, no. 5, pp. 1143–1164, 2008.

[6] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems," in *Proceedings of 20th International Conference on Distributed Computing Systems, 2000*, pp. 464–474, IEEE, Taipei, Taiwan, April 2000.

[7] P. Elango and K. Kuppusamy, "Data replication for the distributed database using decision support systems," *International Journal of Computer Applications*, vol. 69, no. 3, pp. 28–39, 2013.

[8] G. Bent, D. Vyvyan, D. Wood, P. Zerfos, and S. Calo, "Distributed policy based access to networked heterogeneous ISR data sources," in *Proceedings of Ground/Air Multi-Sensor Interoperability, Integration, and Networking for Persistent ISR*, vol. 7694, p. 769406, International Society for Optics and Photonics, Orlando, FL, USA, April 2010.

[9] Z. Guessoum, J. P. Briot, N. Faci, and O. Marin, "Towards reliable multi-agent systems: an adaptive replication mechanism," *Multiagent and Grid Systems*, vol. 6, no. 1, pp. 1–24, 2010.

[10] R. Akbarinia, M. Tlili, E. Pacitti, P. Valduriez, and A. A. Lima, "Continuous timestamping for efficient replication management in DHTs," in *Proceedings of International Conference on Data Management in Grid and P2P Systems*, pp. 38–49, Springer, Bilbao, Spain, September 2010.

[11] J. M. Perez, F. García-Carballeira, J. Carretero, A. Calderón, and J. Fernández, "Branch replication scheme: a new model for data replication in large scale data grids," *Future Generation Computer Systems*, vol. 26, no. 1, pp. 12–20, 2010.

[12] M. H. Zack, "The role of decision support systems in an indeterminate world," *Decision Support Systems*, vol. 43, no. 4, pp. 1664–1674, 2007.

[13] M. G. Martinsons and R. M. Davison, "Strategic decision making and support systems: comparing American, Japanese and Chinese management," *Decision Support Systems*, vol. 43, no. 1, pp. 284–300, 2007.

[14] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: near-optimal block placement and replication in distributed caches," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 184–195, 2009.

[15] H. Mühleisen, T. Walther, and R. Tolksdorf, "Data location optimization for a self-organized distributed storage system," in *Proceedings of 2011 Third World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pp. 176–182, IEEE, Salamanca, Spain, October 2011.

[16] M. Patiño-Martinez, R. Jiménez-Peris, B. Kemme, and G. Alonso, "MIDDLE-R: consistent database replication at the middleware level," *ACM Transactions on Computer Systems*, vol. 23, no. 4, pp. 375–423, 2005.

[17] F. Chang, J. Dean, S. Ghemawat et al., "Bigtable: a distributed storage system for structured data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1–26, 2008.

[18] Y. Elouerkhaoui, "Static replication," in *Credit Correlation*, pp. 195–202, Palgrave Macmillan, Basingstoke, UK, 2017.

[19] E. Derman and N. N. Taleb, "The illusions of dynamic replication," *Quantitative Finance*, vol. 5, no. 4, pp. 323–326, 2005.

[20] L. M. Khanli, A. Isazadeh, and T. N. Shishavan, "PHFS: a dynamic replication method, to decrease access latency in the multi-tier data grid," *Future Generation Computer Systems*, vol. 27, no. 3, pp. 233–244, 2011.

[21] K. Ranganathan and I. Foster, "Identifying dynamic replication strategies for a high-performance data grid," in *Proceedings of International Workshop on Grid Computing*, pp. 75–86, Springer, Denver, CO, USA, November 2001.

[22] M. L. Yiu, H. Lu, N. Mamoulis, and M. Vaitis, "Ranking spatial data by quality preferences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 3, pp. 433–446, 2011.

[23] P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger, "File-based replica management," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 115–123, 2005.

[24] B. Lin, S. Li, X. Liao, Q. Wu, and S. Yang, "eStor: energy efficient and resilient data center storage," in *Proceedings of 2011 International Conference on Cloud and Service Computing (CSC)*, pp. 366–371, IEEE, Hong Kong, China, December 2011.

[25] R. S. Chang, H. P. Chang, and Y. T. Wang, "A dynamic weighted data replication strategy in data grids," in *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, 2008. AICCSA 2008*, pp. 414–421, IEEE, Doha, Qatar, March 2008.

[26] X. Dong, T. El-Gorashi, and J. M. Elmirghani, "Green IP over WDM networks with data centers," *Journal of Lightwave Technology*, vol. 29, no. 12, pp. 1861–1880, 2011.

[27] F. Ping, X. Li, C. McConnell, R. Vabbalareddy, and J. H. Hwang, "Towards optimal data replication across data centers," in *Proceedings of 2011 31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 66–71, IEEE, Minneapolis, MN, USA, June 2011.

[28] W. Li, Y. Yang, and D. Yuan, "A novel cost-effective dynamic data replication strategy for reliability in cloud data centres," in *Proceedings of 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pp. 496–502, IEEE, Sydney, Australia, December 2011.

[29] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "CDRM: a cost-effective dynamic replication management scheme for cloud storage cluster," in *Proceedings of 2010 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 188–196, IEEE, Heraklion, Greece, September 2010.

[30] Y. Qu and N. Xiong, "RFH: a resilient, fault-tolerant and high-efficient replication algorithm for distributed cloud storage," in *Proceedings of 2012 41st International Conference on Parallel Processing (ICPP)*, pp. 520–529, IEEE, Pittsburgh, PA, USA, September 2012.

[31] Y. F. Lin, J. J. Wu, and P. Liu, "A list-based strategy for optimal replica placement in data grid systems," in *Proceedings of 37th International Conference on Parallel Processing, 2008. ICPP'08*, pp. 198–205, IEEE, Portland, Oregon, September 2008.

[32] V. Andronikou, K. Mamouras, K. Tserpes, D. Kyriazis, and T. Varvarigou, "Dynamic QoS-aware data replication in grid environments based on data "importance"," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 544–553, 2012.

[33] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, pp. 205–216, ACM, Indianapolis, IN, USA, June 2010.

[34] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya, "Energy-efficient data replication in cloud computing datacenters," *Cluster computing*, vol. 18, no. 1, pp. 385–402, 2015.

[35] Y. Mansouri, A. N. Toosi, and R. Buyya, "Cost optimization for dynamic replication and migration of data in cloud data centers," *IEEE Transactions on Cloud Computing*, 2017.

[36] V. Nagarajan and M. A. M. Mohamed, "A prediction-based dynamic replication strategy for data-intensive applications," *Computers & Electrical Engineering*, vol. 57, pp. 281–293, 2017.

[37] S. Pan, L. Xiong, Z. Xu, Y. Chong, and Q. Meng, "A dynamic replication management strategy in distributed GIS," *Computers & Geosciences*, vol. 112, pp. 1–8, 2018.

[38] E. Brewer, "A certain freedom: thoughts on the CAP theorem," in *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, p. 335 ACM, Zurich, Switzerland, July 2010.

[39] S. Gilbert and N. Lynch, "Perspectives on the CAP theorem," *Computer*, vol. 45, no. 2, pp. 30–36, 2012.

[40] B. W. Diack, S. Ndiaye, and Y. Slimani, "CAP theorem between claims and misunderstandings: what is to be sacrificed," *International Journal of Advanced Science and Technology*, vol. 56, pp. 1–12, 2013.

[41] M. Kleppmann, "A critique of the CAP theorem," 2015, http://arxiv.org/abs/1509.05393.

[42] O. Patinge, V. Karkhanis, and A. Barapatre, "Inadequacies of CAP theorem," *International Journal of Computer Applications*, vol. 151, no. 10, pp. 18–20, 2016.

[43] N. K. Gill and S. Singh, "A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers," *Future Generation Computer Systems*, vol. 65, pp. 10–32, 2016.

[44] L. Wujuan and B. Veeravalli, "An object replication algorithm for real-time distributed databases," *Distributed and Parallel Databases*, vol. 19, no. 2-3, pp. 125–146, 2006.

[45] B. K. Panigrahi, Y. Shi, and M. H. Lim, *Handbook of Swarm Intelligence: Concepts, Principles and Applications*, Springer Science and Business Media, Vol. 8, Springer Science and Business Media, Berlin, Germany, 2011.