Mathematical and Computational Applications

MDPI

Article

The Impact of the Implementation Cost of Replication in Data Grid Job Scheduling

Babar Nazir [1,*], Faiza Ishaq [1], Shahaboddin Shamshirband [2,3] and Anthony T. Chronopoulos [4,5]

1 Department of Computer Science, COMSATS University, University Road, Tobe Camp, Abbottabad 22060, Pakistan; faizaishaq27@gmail.com
2 Department for Management of Science and Technology Development, Ton Duc Thang University, Ho Chi Minh City, Vietnam; shahaboddin.shamshirband@tdt.edu.vn
3 Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam
4 Department of Computer Science, University of Texas at San Antonio, San Antonio, TX 78249, USA; antony.tc@gmail.com
5 (Visiting Faculty) Department of Computer Engineering & Informatics, University of Patras, 26504 Rio, Greece
* Correspondence: babarnazir@gmail.com

Received: 11 May 2018; Accepted: 16 May 2018; Published: 25 May 2018

Abstract: Data Grids deal with geographically-distributed large-scale data-intensive applications. Schemes scheduled for data grids attempt to not only improve data access time, but also aim to improve the ratio of data availability to a node, where the data requests are generated. Data replication techniques manage large data by storing a number of data files efficiently. In this paper, we propose centralized dynamic scheduling strategy-replica placement strategies (CDSS-RPS). CDSS-RPS schedule the data and task so that it minimizes the implementation cost and data transfer time. CDSS-RPS consists of two algorithms, namely (a) centralized dynamic scheduling (CDS) and (b) replica placement strategy (RPS). CDS considers the computing capacity of a node and finds an appropriate location for the job. RPS attempts to improve file access time by using replication on the basis of number of accesses, storage capacity of a computing node, and response time of a requested file. Extensive simulations are carried out to demonstrate the effectiveness of the proposed strategy. Simulation results demonstrate that the replication and scheduling strategies improve the implementation cost and average access time significantly.

Keywords: data grid; replica placement; replication; scheduling

1. Introduction

The term *grid computing* was first put forth by Foster and Kesselman [1,2] as the paradigm that provides reliable, consistent resource sharing and execution of jobs in the distributed systems. A data grid is an integrated architecture that offers a collection of storage and computational resources in distributed geographical locations [3]. Data grids involve the management and storage of huge amounts of data. Key data grid application areas include (a) climate simulation, (b) the Large Hadron Collider (LHC), (c) astronomy and earth observation, and (d) high-energy physics (HEP) that produce a huge volume of data (terabytes to petabytes) [4–11]. Data grid scheduling strategies should attempt to optimize the data availability, job execution time, fault tolerance, bandwidth consumption, and response time [12].

Techniques that are applied to overcome the data availability problem are (a) caching, (b) mirroring, and (c) replication. The cache is small in size and stores the most commonly requested data, but in large computing environments a small cache may not fulfil the data request rate for each computational resource [13]. Mirroring conveys the data's meaning, coping files from one server to other. Mirroring is not a practical technique, as it needs to move all of the data from one site to the other all the time, as the request rate changes periodically. Replication involves deploying several data files on servers to fulfil the client request more rapidly. Replication normally increases the accessibility of data and optimizes response time and bandwidth consumption rate, which ultimately boosts the system performances. Considering the dynamic data grid environment that involves managing huge amounts of data, a replication technique is considered in order to improve data access time in a data grid environment.

Data replication when employed in data grids improves efficiency and reliability in dealing with wide area data accessing and retrieving processes [14,15]. Data replication techniques are of two types: (a) static and (b) dynamic [16]. In static replication techniques, a number of created replicas occur within the same node, until the user deletes them or time duration expires. In dynamic replication techniques, data files are duplicated based on user requests and storage capacity requirements of a computing node.

An important issue in data replication techniques is replica placement [17]. Replica placement is the assignment of the duplicate copies of data to an appropriate number of nodes. Replica placement is further categorized into two phases: (a) replica discovery and (b) replica selection. Replica discovery decides when and where to place the replica. Replica selection decides which file to replicate [18,19]. If sufficient storage space is not available in a node then the file replacement stage is activated for new replicas [8]. Thus, the best replica placement strategy improves the response time and bandwidth consumption significantly.

The motivation of this paper is to develop a scheduling and replication strategy that not only minimizes the implementation cost, but also efficiently schedules the tasks with minimum data movements. The scheduling strategy should be able to minimize the transfer and deletion actions of data files among computing nodes. Devising such a strategy is a challenge, as users' request patterns may change very frequently in a data grid environment. Such an example is a video server system, where the demand for videos changes each day. Therefore, fulfilling the required demand from the nearest server by replicating and replacing new files all the time is not an appropriate methodology. Scheduling jobs for an appropriate computing node is essential, since data transferring among computing nodes increases the implementation cost and data access time [2]. Therefore, in our proposed strategy, we have considered both the replication and job scheduling.

In this paper, a centralized dynamic scheduling strategy-replica placement strategy (CDSS-RPS) is proposed. CDSS-RPS consists of two strategies: (a) centralized dynamic scheduling strategy (CDSS) and (b) replica placement strategy (RPS). CDSS considers the data grid architecture that supports effective scheduling of job and data replication. RPS replaces replicas on the computing sites that have enough storage space and computing capability. We have considered the replication and scheduling strategy equally to minimize the job execution time.

The main contributions of this paper are as follows:

1. We present a job scheduling strategy for data-intensive applications in a data grid environment. Our paper modifies the cost minimization strategy presented in [19], and considers the replication strategy along with the scheduling strategy.
2. We present a novel data replication and scheduling strategy. Our approach consists of minimizing the data access time and implementation cost, i.e., (a) scheduling the tasks on the available computing node for the number of tasks requested, and (b) minimizing the transfer time of all data files with respect to their network bandwidth consumption.
3. GridSim Tool Kit 4.1 [20] was also used to define the data grid architecture that supports job scheduling and data replications and perform the simulation.

4. Extensive simulations were carried out to compare the implementation cost and turnaround time of the proposed strategy with the contemporary data replication strategies in a data grid environment.

The rest of the paper is organized as follows: Section 2 discusses the related work. In Section 3, the proposed work is described, i.e., the system model, the problem formulation, and the proposed methodology. The methodology experimentally evaluated and results are presented in Section 4. Finally, Section 5 concludes the paper.

2. Related Work

The aim of dynamic replication and scheduling strategy is to minimize the cost of accessing the data and the execution time of a job in a data grid environment. The scientific applications consist of huge sets of data files in a data grid environment. Due to storage constraints of computing sites, all the files cannot be stored on each and every computing node. An early work on dynamic data replication in grids is presented by [21,22], where the following replication strategies have been proposed: (a) no replication, or plain; (b) cascading; (c) best client; (d) cascading plus caching; and (e) fast spread. The study concludes that fast spread shows consistent performance through various access patterns. The modified version of the fast spread replication strategy is presented by Bsoul et al. [23]. They proposed a dynamically-based replication strategy and presented results demonstrating that it is better than the fast spread strategy.

Replica placement has been considered in several circumstances, e.g., web servers, video, and content distribution networks (CDNs) [24,25]. The replica placement problem (RPP), also called the file allocation problem [26], considers the data management. Moreover, RPP presumes only a fixed system state, and cannot manage the frequent changes of a replication scheme. In Loukopoulos et al.'s work [27], the continuous replica placement problem (CRPP) considers frequent updating requirements, but the main drawback of CRPP is that it does not consider the cost factor. In [19], the replica transfer scheduling problem (RTSP) not only considers the frequent replication of data, but also handles the cost minimization problem while transferring data from one state to another. RTSP does not consider the scheduling of job and task in order to fulfil the request within a time deadline.

Maheswaran et al. [28] introduced the heuristics of matching and scheduling the independent jobs of different types of classes. These heuristics are classified into two modes: (a) the online mode and (b) the batch mode. Hamscher et al. [29] considered the structure of a scheduling process and assumed all data structures, including centralised, hierarchical, and decentralized, for the grid environment. The result is not meant to be complete, as the scheduling algorithm is dependent upon the parameters and machine configuration, and the work is done on the limited number of parameters. Shan and Oliker [30] suggest super-scheduler architecture and three algorithms that are based on the distributed job migration algorithms. These studies do not consider the control and the proper management of data replicas and scheduling jobs properly.

Park et al. [31] presented a bandwidth hierarchy-based replication (BHR) that is used to reduce the time of accessing data and avoid any blockage of a network, by enhancing network-level locality. The BHR technique works well in cases where the storage capacity is considered to be smaller, but BHR has two main shortcomings: (a) if a region contains the replica, it terminates; and (b) instead of selecting a few appropriate sites, replicas are placed in all the demanded sites.

Chang et al. [32] proposed an emergent methodology called fragmented replicas. The approach replicates only that part of the file that needs to be replicated, in order to save storage space. The proposed fragmented replica strategy still needs to be improved. Moreover, the assumption on fragmented replicas being adjacent might not be easily possible in a data grid environment. In addition, the authors have proposed the hierarchical cluster scheduling strategy (HCS) and the hierarchical replication strategy (HRS) in [33–36], to improve the efficiency rate of accessing the data in the grid. HCS uses hierarchical scheduling and involves the information of a cluster to reduce the time searching for the appropriate computing node. HRS uses the same idea of "network locality" as the BHR strategy. The HRS technique has two benefits over BHR: (a) BHR discovers the best replica by checking all the sites, while in the case of HRS a replica is located in a local cluster; and (b) HRS characterizes the replica at the site level, while BHR characterizes the replica at the cluster level. The HCS and HRS strategy improve the time it takes to access the data and perform inter-cluster communications, in contrast to other scheduling algorithms and replication strategies. However, HRS focuses on deleting and selecting the replica on the basis of bandwidth. An extension of the BHR [31] strategy is the customized and the modified BHR [36] strategy, which replicates the most accessed data files that are supposed to be requested again in a near future. The modified BHR strategy has a major drawback in that it replicates the file to the best site by searching all the sites individually.

In [22], both a three-level hierarchy structure and a scheduling algorithm are proposed. The work defines a network hierarchical structure that has three levels. The three-level hierarchy technique leads to an improved performance over the LRU (least recently used) technique. Andronikou et al. [37] project novel data replication strategies that consider the infrastructural controls as well as the "significance" of the data. In this work, a QoS-aware dynamic replication strategy concluded the amount of replicas needed based on content importance, data request, and requested QoS. The QoS-aware dynamic replication strategy places the upcoming or new replicas based on the network bandwidth capacity and the replication strategies within the grid environment. The dynamic replication strategy also considers the grid environment, so as to increase or decrease the number of replicas on the computing sites based on the data request rate. Taheri et al. [38] proposed a bee colony-based optimization algorithm, calling it job data scheduling using bee colony (JDS-BC). JDS-BC actually consists of two methods: files replication on computing sites and job scheduling onto computational sites in a system. This method reduces the total time to transfer the files in a varied system environment, and is capable of working under several scenarios.

Vrbsky et al. [39] focused on an energy-efficient data-aware strategy, and presented a new data replication strategy called "queued least frequently used" (QLFU). QLFU measures replication performance and determines if it reduces energy consumption and ultimately saves energy significantly, without any effect on the response time. Almuttairi et al. [40] designed a replica selection strategy called "two-phased service-oriented broker" (2SOB). 2SOB consists of two phases: (a) coarse-grain phase and (b) fine-grain phase. Replica placement to the site that has an uncongested network link from the congested link is done in first phase of 2SOB. The fine-grain phase is designed to fulfil the user request as quickly as possible and with the minimum network cost. Both phases are designed and simulated in the European Data Grid. Gui et al. [41] proposed an algorithm based on replica selection. A scheduling algorithm along with replica selection is also proposed. The three scheduling algorithms are the global dynamic scheduling algorithm (GDSA), the local dynamic scheduling algorithm (LDSA), and the naive scheduling algorithm (NSA). The main objective of these algorithms is to improve data latency time and network bandwidth consumption.

Mansouri et al. [16] proposed two strategies. First is weighted scheduling strategy (WSS), which is a scheduling strategy that aims to reduce the search time for a suitable computing site. WSS examines the sets of requested jobs waiting for their turn in a queue, the place to where the requested data should be replicated, and the storage capacity of a computing site. The second is a dynamic data replication strategy called "enhanced dynamic hierarchical replication" (EDHR), which advances file access time. EDHR strategy is an enhanced version of the dynamic hierarchical replication strategy. EDHR introduces a method to delete the files from the computing nodes whenever there is not enough space for the new. An ideal replica placement strategy improves the job execution time and the cost to store data.

Li et al. [42] proposed the cost-effective data-reliable PRCR mechanism, based on a data-reliability model, using a proactive replica-checking approach. This approach manages the minimum required number of replicas, in order to reduce the replication cost, and manages large amount of data in the cloud at the minimum cost of cloud storage space. This strategy ignores the location of the replicas in the cloud.

Souravlas et al. [43] proposed a technique using a binary tree structure to handle the increasing and decreasing demands of the user over a period of time in each node of the data grid computation. The proposed technique calculates the number of files to be accessed more frequently, based on the temporal locality principle. The strategy improves the completion of file access time performances requested by the user.

Bsoul et al. [17] have also worked on the same idea to improve the data replication strategy. In order to select the best file to be replicated, they proposed the round-based data replication strategy. The simulation result shows that this strategy improves the performance in terms of average number of files delayed and the average of bandwidth consumption.

Liu et al. [44] presented SD^3 (selective data replication in distributed datacenters) mechanism for OSN (online social networks), with the aim of reducing the inter-datacenter data communications within the limits of achieving low service latency.

Michelon et al. [45] proposed data replication strategy MANETs (mobile ad hoc networks) to improve data access for mobile ad hoc networks. MANETs aim to solve the problem of handling a series of application-dependent requirements through an application-level message forwarding mechanism using virtual magnetic fields.

Nahir et al. [46] present a replication-based load balancing mechanism over large distributed server systems. The approach was based on creating multiple replicas of each job and sending them to the different servers; when each replica arrives at the head of the queue at its server, then signals will be sent to rest of the servers to remove that job from their queues. The proposed scheme not only overcomes the overhead communication between the users and the servers, but also removes the overhead scheduling from the job's critical path.

Souli-Jbali et al. [47] present a dynamic data replication-driven model that, in order to best exploit the available resources, combines scheduling queries and data replication, which also minimizes the total cost. Patil et al. [48] work on directory server data replication to improve data replication results. The authors propose a mechanism of replication between heterogeneous supported platforms through service-oriented architecture.

Khalajzadeh et al. [49], by using graph partitioning, present a cost efficient data placement and a replication technique that finds near-optimal data placement of replicas, and minimizes the monetary cost as well. Usman et al. [50] present a component-based data replication strategy, CbHaRS (component-based highly-available replication strategy), that achieves high operation availabilities through hybrid communication model, and also exploits the operation types.

Mansouri [51] enhanced the data grid by proposing threshold-based dynamic data replication (TDDR), a job scheduling strategy, with parallel job scheduling, which schedule jobs and manage data at the same time. Zeng et al. [52] present a data management and task computation scheduling strategy ADAS (adaptive data-aware scheduling) for workflow applications, which achieves a satisfactory makespan through data management and reveals parallelization mechanism. Casas et al. [53] propose a balanced scheduler technique for scientific workflows with data reuse and replication, balanced and file reuse–replication scheduling (BaRRS), which optimally schedules scientific application workflows.

A survey on data replication strategies in P2P systems was conducted by Spaho et al. [54], in which data replication strategies and requirements for various P2P application were identified. The authors also classify the replication control mechanisms on the basis of "where, when, and how" the updates are take place, propagated and distributed fully or partially. Another survey regarding the trends in data replication strategies was conducted by Souravlas et al. [55]. The authors identified various data replication techniques, including both static and dynamic strategies, and also mentioned the advantages and disadvantages of each technique.

After having gone through an extensive literature review, we are of the opinion that the research presented in this paper is closely related to the aspects mentioned above, in which data replication and job scheduling strategies are combined. Compared with the existing strategies, our strategy is more comprehensive, in not only reducing job execution time but also minimizing the implementation cost.

Motivated by this, we propose the CDSS (centralized dynamic scheduling strategy) for the scheduling of jobs, and RPS (replica placement strategy) for replica placement. The proposed data replication and job scheduling strategy attempt to reduce job execution time, as well as decrease implementation cost, by minimizing the data transfer and deletion rate in a data grid environment. Moreover, the proposed strategy ultimately minimizes implementation cost and improves the data access rate significantly.

2.1. Implementation of Cost Minimization Techniques

The replica transfer scheduling [19,56] technique is exclusively proposed for minimizing the cost of transferring the data from one computing site to another. XOLD and XNEW are two input states—an old and a new state, respectively—and produce a valid schedule H (the number of files record in a history table) for migrating from XOLD to XNEW. The following algorithm tries to minimize the implementation costs.

2.1.1. All Random

The aim of this algorithm is to create replicas requested by a user in a new state and delete the replicas that are present in an old state. This decision is taken randomly. Outstanding replicas are those that are required in the XNEW state, but are not available in the XOLD state. Whereas superfluous replicas are those that are not required in the XNEW state, but are available in the XOLD state. All random (AR) does not consider the optimization of total communication cost.

2.1.2. Highest Opportunity Cost First

The highest opportunity cost first (HOCF) technique is specifically designed to minimize the implementation cost by selecting the outstanding replicas. The cost of the nearest source node is given as equal to 0, and the servers that are next to the nearest node are prioritized according to the implementation cost. If the nearest source server is deleted, then the requested replica is transferred from the second nearest source node. Every iteration selects the outstanding replicas that have the highest opportunity cost. If not enough space is available for replicas, one or more replicas are deleted from the target server. Outstanding replicas having zero or less opportunity cost are selected at the end in random order.

2.1.3. Greedy Object Lowest Cost First

The greedy object lowest cost first (GOLCF) heuristic technique is different from HOCF. In this technique, an object is selected randomly, to be replicated on all those servers that require a copy of it. A request of a source node is fulfilled from the nearest node having the lowest communication cost. If not enough space is available, then one or more replicas need to be deleted. The deletion action is performed very efficiently so that no usable file is deleted. The process is repeated for all the objects that are under consideration. When no more objects are left to be considered, the algorithm terminates.

GOLCF focuses on replicating the objects when they need to be replicated, and completes objects set to be replicated at a given time. Replicating all the objects helps optimize the sequence of the resulting transfers, whereas the random transfer of objects may lead to non-optimal overall results.

3. Proposed Work

The following subsection explains the system model, the working of CDSS-RPS, the CDSS-RPS, and the architectural diagram in detail.

3.1. System Model

In a data grid system, efficient network performance is required to schedule and dispatch the number of jobs required by the end user. Figure 1 shows how a data grid system supports, schedules, and dispatches the jobs. A typical data grid system is comprised of a set of domains that consist of a replica server and a number of computing sites. Storage for replicas and computational resources for the jobs are provided by the replica server and the computing site, respectively. Both of these functioning devices are commonly known as nodes. A local area network (LAN) serves every two nodes within domains with a similar network bandwidth. Also, a wide area network (WAN) connects different domains' nodes with different network bandwidth.

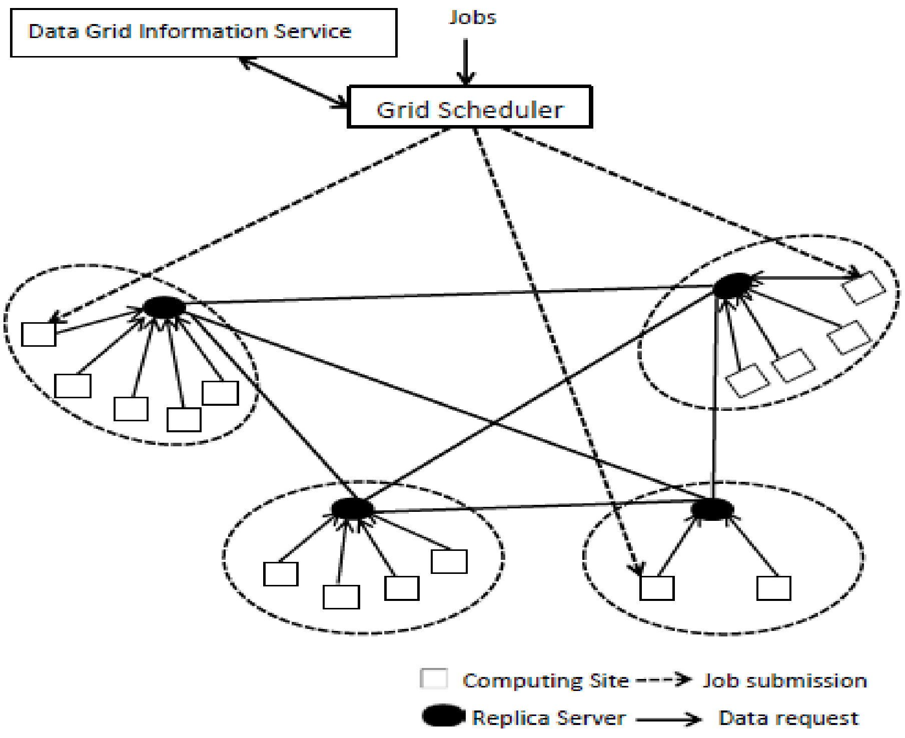


Figure 1. System architecture.

If a replica server exists in the same domain as a computing site, it is known as the primary replica server (PRS) for that computing site, while all other replica servers are considered to be secondary replica servers (SRS). The data grid information service keeps a record of available resources and provides resource registration service in the data grid. Based on specific strategies, it is the job of the grid scheduler to allocate a task to the specific computing nodes, as shown in Figure 1. A local storage device stores the data before it is allocated to the computing nodes. If the cache of a computing node does not contain the required data, then the request for the data will be directed to its PRS. PRS will search within a domain first, but if the required data is not available within a domain, then PRS will search among all replica servers and collect the data from the replica server having the highest bandwidth capacity connection to the computing node. The ideal grid scheduler should assign the jobs to appropriate computing sites with all the available data files, which ultimately minimizes the search time significantly.

The data grid architecture is defined as follows:

- There are D domains in the data grid.
- For every domain $d \in D$, the number of computing sites situated in d is denoted by CS(d).
- RS(d) are the replica servers within each domain d.
- NS(d) is the set of nodes within each d, where NS(d) = CS(d) U RS(d).
- C_i is the computing capability for a computing sites i.

Our CDSS-RPS strategy is based on two phases: CDSS and RPS. In CDSS, the most-accessed jobs are collected from every server in the meta scheduler. The meta scheduler then takes the average of the number of files to be accessed. Then, the RPS strategy is applied to replicate the most-accessed jobs (calculated by CDSS) at the desired computing sites where the demand of the file occurred most frequently. The system architecture with the proposed strategy is explained below.

3.2. Centralised Dynamic Scheduling Strategy-Replica Placement Strategy Architecture Diagram

Figure 2 describes the working of the proposed strategy.
The mechanisms of architecture diagram are described below.

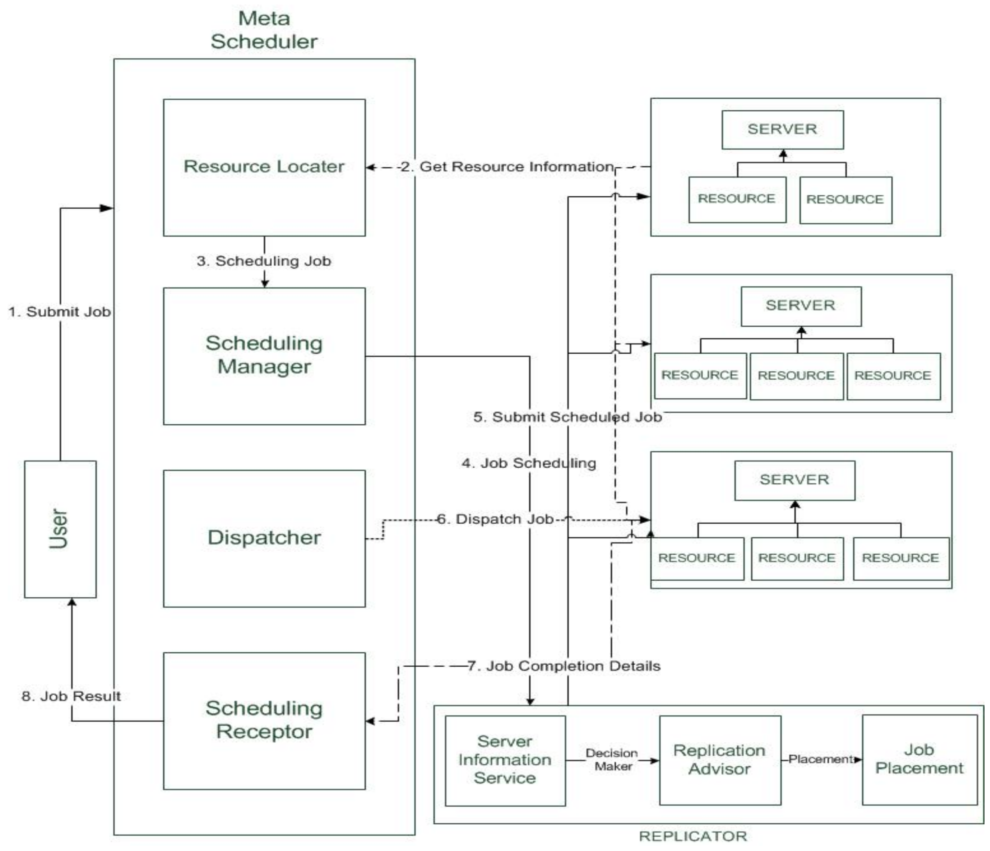


Figure 2. Architecture diagram for centralized dynamic scheduling strategy-replica placement strategies (CDSS-RPS).

3.2.1. Grid User

A grid user submits a job to the meta scheduler by specifying features of jobs (i.e., length of a job, expressed in millions of instructions per second (MIPS)) and quality of service requirements (i.e., processing time).

3.2.2. Meta Scheduler

The resource locator in a meta scheduler registers resources with grid information service (GIS). At the time of registration, the resource locator gets the information of number of computing resources, computing capability, and processing speed from each domain.

3.2.3. Scheduling Manager

The scheduling manager collects the details of available resources and number of user's job requests to execute. The scheduling manager sends details of the number of jobs and available resource information to the replicator for evaluating the list of data files accessed frequently.

3.2.4. Replicator

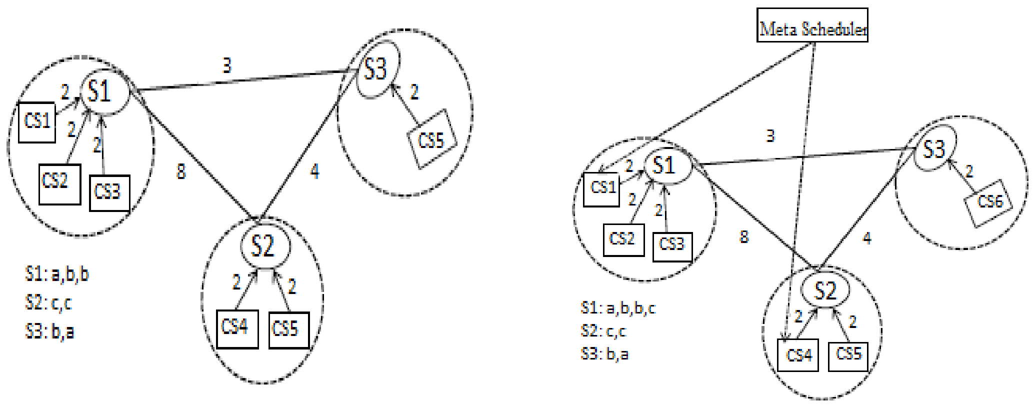
The replicator service consists of three phases: (a) information service, (b) replication advisor, and (c) job placement. The information service gets the information about the number of data files assigned to the computing sites of each domain. The replication advisor then takes the decision on the collected information and invokes the CDSS algorithm discussed in Section 3.2.4.1. The replica placement of a data file and job placement occur as discussed in Sections 3.2.4.2 and 3.2.4.3. The reorganized information about resources and data files is updated on each replica server and the computing sites of each domain.

3.2.4.1. Centralised Dynamic Scheduling Strategy (CDSS)

In a centralized dynamic scheduling strategy, there is a meta scheduler running in a system, elaborated in Figure 3. Every primary replica server (PRS) collects the information of data access from each computing site or resource discussed in Figure 1. PRS sends the number of files and its number of access (NOA) to the meta scheduler. The meta scheduler aggregates the number of accesses of a file and stores the result in a history table H. The centralized dynamic scheduling strategy is invoked by the meta scheduler, which informs the replica server to perform replication. The formula to calculate the average number of access file is as follows.

$$\overline{NOA} = \frac{1}{|H|} \sum_{h \in H} NOA(h) \tag{1}$$

NOA is number of access of each file identification (FID). $|H|$ is the number of files records in a history table H, and $\overline{NOA}$ is the average number of access.



(**a**) (**b**)

Figure 3. Illustrative example (**a**) without scheduler and (**b**) with scheduler.

The replica placement and scheduling strategy is presented in the CDSS algorithm.

Computing sites with a higher computing capability fulfil requests on time or even sometimes before the specified time. The computing capability of domain D is defined as $c_d = (D)c_i$, and the computing capability of all domains is $c_D = c_d$. Let us suppose the relationship between data request rate is proportional to the computing capability. The proportionality difference between them is measured by the factor θ from domain D, and can be denoted by:

$$D\_REQ(D) = \theta \cdot c_d \tag{2}$$

where the θ value should be between 0 and 1. Let us take θ = 1 then $D\_REQ\ (D) = c_d$. For data *f*, where *f* represents the data files, the probability of the requested rate is denoted by Prob_f. Therefore, the data request rate for data *f* from domain *d* can be defined as

$$D\_REQ(D, f) = \text{Prob}_f \cdot D\_REQ(D) = \theta \cdot \text{Prob}_f \cdot c_d \tag{3}$$

Algorithm 1: CDSS Algorithm

```
Input: Number of files to be processed
Output: collect the number of files to be accessed more frequently.
Begin
1: Meta scheduler collects the number of accessed files information from each replica server.
2: Meta scheduler aggregates the number of access of files.
3: The average number of accesses is calculated according to Equation (1).
4: For each file from the history table do
5:     Compare each file access with the average number of accesses.
6:     if
7:         the number of file accesses is less than the average number of accesses
8:         Remove files accessed
9:     End if
10:    Sort the remaining files in descending order.
11:    Last record in a history table is denoted by LAF, where LAF is the least accessed file.
12: End for
13: Place the files on replica servers following the history table $H$.
14: While history table $H$! = empty
15:    Pop h from the history table $H$
16:    Request from replica placement strategy (RPS) (Algorithm 2) to replicate the file id (FID) (h).
17:    Update record $h$
18:        If $NOA$ ($h$) > LAF then
19:            Re-insert record $h$ in $H$ according to descending order of $NOA$ field
20:        End if
21: End while
End
```

The computing node, having the highest available bandwidth, most likely has the capability to directly access the replicas. The bandwidth capacity between computing nodes within the same domain is supposed to be the same. $\text{BW}_{d,k}$ is a measure of the bandwidth capacity between the computing node and replica server *k*, where a computing node can be any node in domain *d*. The number of replica servers in data grid architecture is denoted as R_SER, which stores the original data, or the replicas of data *f*. The bandwidth capacity B_CAP while accessing the data *f* from any node in domain *d* can be defined as

$$\text{B\_CAP}(d, f) = \max_k \in \text{R\_SER}_f \text{BW}_{d,k} \tag{4}$$

The average response time for data *f* is defined as follows

$$\text{AvgResp Time}(f) = \sum_{d \in D} \frac{c_d}{\max_k \in \text{R\_SER}_f \text{BW}_{d,k}} \tag{5}$$

The average response time is the minimal average response time for any data *f*.

The replica placement is made on the basis of largest bandwidth capacity and storage place of every replica server. Let R_f be the set of servers containing replicas or the original copy of data *f*. Replicate the data *f* to the replica server, where data *f* does not exist and has enough storage space to replicate data.

3.2.4.2. Replica Placement Strategy

Let XNEW contain the set of data files that are requested in a new session, and XOLD contain the set of replicas in an old session. After scheduling and placement of data file *f*, a data request is generated on a number of replica servers. These requests take replica placements XOLD and XNEW as inputs, and introduce additional (superfluous) transfer actions as early as possible to the start of data request submitted. Superfluous replicas are defined as the data files that are not present in the XNEW state, but were available in XOLD state. Corresponding deletion action is added to schedule before the transfer action takes place. The next session explains the transfer and deletion mechanism, as well as how to produce a new valid schedule that incurs the lowest cost.

Algorithm 2: RPS Algorithm

```
Input: Most accessed data files that need to be replicated.
Output: Replicate the files on the computing sites
Begin
1: Calculate the computing capability of a domain $c_d$ and sum of all domains $c_D$.
   Where $c_d = (D)c_i$ ($D$ is the domain)
   And $c_D = c_d$ (computing capability of all domains)
2: Data request rate is proportional to its computing capability; proportional relationship is presented in
   Equations (2) and (3).
3: Set bandwidth equal between computing nodes within the domain and varying outside the domain.
4: Calculate the highest available bandwidth between the nodes in domain $d$ to replica server $k$ as shown in
   Equation (4).
5: Average response time for most accessed or requested data files are calculated in Equation (5).
6:     If bandwidth capacity and storage capacity is greater than that of the file to be replicated do
7:         If else replica server does not already contain that data file do
8:             Replica placement
9:         End if else
10:    Else
11:        Delete data files that are not required in a new requested state.
12:    End if
End
```

3.2.4.3. Schedule Enhancement Operator

When the most accessed data files are replicated over the number of servers, the request is generated to check the cost of transfers and deletions. In case the requested data file is inaccessible from the host server, it is transferred from the computing site having the highest available bandwidth and it is replicated on the host server as well. We next present Algorithm 3, which considers the bandwidth as a main feature for a selection and deletion of a replica. If the storage size of a server is smaller, then the algorithm deletes the files that are not required in a new state but were available in an old state, or the files that are least recently used.

3.2.5. Dispatcher

The dispatcher dispatches the tasks one by one from the queue to the computing sites for which they are scheduled.

3.2.6. Scheduling Receptor

The scheduling receptor gets the task completion details from each computing sites and sends the completion results details to the user.

Algorithm 3: Schedule Enhancement Operator

```
Input: Request is generated
Output: Request is granted successfully
Begin
1: If (requested replica is in a computing site) $R_r = C_s$
2:     fulfill the request
3: Else if (requested replica is not in a computing site) $R_r ! = C_s$
4:     search replica within the same domain Goto domain $C_{d(i)}$
5:         If (replica is found in the same domain) $R_r = C_d$
6:             move to requested site $R_r = C_s$
7:                 If (! Enough space available) (space! = 0)
8:                     Directly fulfill the request    // avoid duplication
9:                 End if
10:        else
11:            search replica in other domain $Goto\ (D)c_i$ ($D$ is the domain)
12:            create a list of computing sites that offer requested replica
13:            select the replica from the computing site that offers the highest available bandwidth
14:        End else
15:        End if
16:            If (space is available to store new replica)
17:                store it
18:            Else
19:                while (not enough space is available)
20:                    delete least accessible files
21:                    store new replica
22:                End while
23:            End else
24:            End if
25: End else if
26: End if
End
```

3.3. Illustrative Example

For better understanding, we choose to provide a complete scenario, to elaborate the working of the proposed strategy illustrated via a simple example, shown in Figure 3. Figure 3a,b show the difference between the implementation costs without scheduling the data files on each domain and with a scheduler, defining the job submission and replication. Suppose there are three replica servers, each in a domain. CS_i are computing sites connected to each replica server with the same bandwidth within a domain. The network bandwidth between different replica servers is different. Suppose S_1 consists of data files (a, b, b), S_2 consists of data files (c, c), and S_3 contains data files (b, a). The network bandwidth between S_1 and S_2 is 8 bps, 4 bps between S_2 and S_3, and 3 bps between S_1 and S_3. The network bandwidth between the computing sites and the replica server is supposed to be the same within a domain.

In Figure 3a, the request is submitted to any computing site. Let us suppose that a job is submitted to CS_1, and the job requires data file c. S_1 does not contain data file c, therefore it is transferred from S_2 to fulfill the request of a job. If S_1 does not contain enough space, it will delete any file to place replica c. Similarly, the same situation occurs if b is requested in S_2; this also requires the transfer and deletions to fulfill the request. Transfers and deletions increase the implementation cost, delay data request completion, and cause congestion in network traffic. Figure 3b explains the same situation with the meta scheduler. The meta scheduler holds all the information of each domain and decides the job submission on its base. It replicates the most-accessed data files to the number of computing sites, wherever the data request rate is higher than others. In Figure 3b, the meta scheduler submits the jobs to S_2 directly, minimizing the implementation cost and ultimately decreasing data access time as well. This shows that adding the scheduler significantly improves the efficiency of accomplishing the user requests.

4. Simulation Results and Discussion

The following subsection explains the simulation results and their discussion in detail. It includes details of the simulation setup, performance evaluation parameters, results, and discussion of those results.

4.1. Simulation Setup

To evaluate the effectiveness of the proposed strategy, we carried out a simulation using the Grid Sim simulator [20,57]. For simulation in a data grid environment, we considered 25 sets of domains, and each and every domain had a server, also called the replica server. We took 80 computing sites that were randomly dispersed to the 25 domains. The links between computing site nodes and server nodes were given a bandwidth that was randomly distributed. Within the domain, the bandwidth was set to be within the range of 15 to 100 Mbps, but outside the domain the bandwidth was set normally to less than 15 Mbps. The number of files differed from 100 to 500, and the size of the files was homogeneously dispersed from 100 to 400. The primary replicas were assigned randomly to all the nodes.

We compared the performance of the proposed strategy, GOLCF, with the contemporary strategy, HOCF, as these strategies are more related to our work as compared to rest of the newer strategies mentioned in related work. The performance evaluation parameters that were used to compare results include implementation cost, i.e., the cost associated with implementing the data grid; geometric mean of turnaround time; turnaround time; and average turnaround time.

4.2. Results and Discussion

Simulation results, performance evaluation parameters, and their discussions are presented in this section.

4.2.1. Evaluation of Replica Transfer Scheduling Problem Algorithms

Implementation cost was compared initially with the schedules that were produced by non-scheduled RTSP algorithms. Figure 4 shows the performance of the schedules created by GOLCF and HOCF from [19]. The number on the x-axis represents the replicas per object created, and the number on the y-axis represents the implementation cost. Let us suppose that XNEW changes with the replica numbers need to be created for each object. Storage capacity is supposed to be kept equal for all the replicas in XNEW.

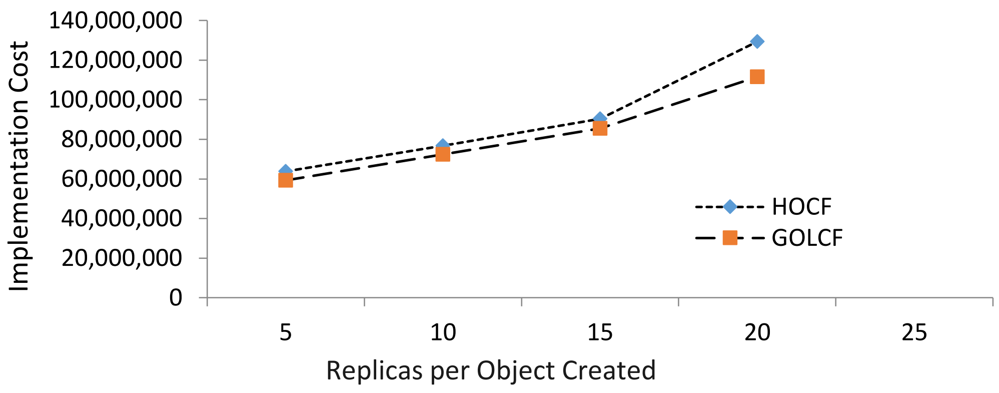


Figure 4. Cost of replica transfer scheduling problem (RTS)-based algorithms.

4.2.2. Comparison of Centralized Dynamic Scheduling Strategy-Replica Placement Strategy with Greedy Object Lowest Cost First and Highest Opportunity Cost First

For a performance comparison with the CDSS-RPS, we chose the best contemporary algorithms, which include GOLCF and HOCF CDSS-RPS, and RTSP algorithms, which were taken as input for the XOLD and XNEW states. Figure 5 represents the results for the implementation cost for CDSS-RPS, GOLCF, and HOCF, respectively, when the replica per file size varied from 5 to 100. The number on the x-axis represents the replicas per object created, and the y-axis represents the implementation cost. Implementation cost is based on the number of transfers and deletions occurring in order to fulfill the requests. When the number of transfers increases, it also increases the implementation cost. Figure 5 shows that the implementation cost of our strategy is less than the other two. CDSS-RPS is a better choice compared to GOLCH and HOCF. GOLCF makes the decision on the basis of the smallest link cost, and gives better results as compared to HOCF, but the number of files and jobs in GOLCF are not scheduled properly, as it does not consider the most-accessed files and their replications. The HOCF strategy is based on the nearest and second-nearest goal server. CDSS-RPS schedules the most accessed files, then replicates the files on its access bases; this improves the implementation cost significantly, as shown in Figure 5.

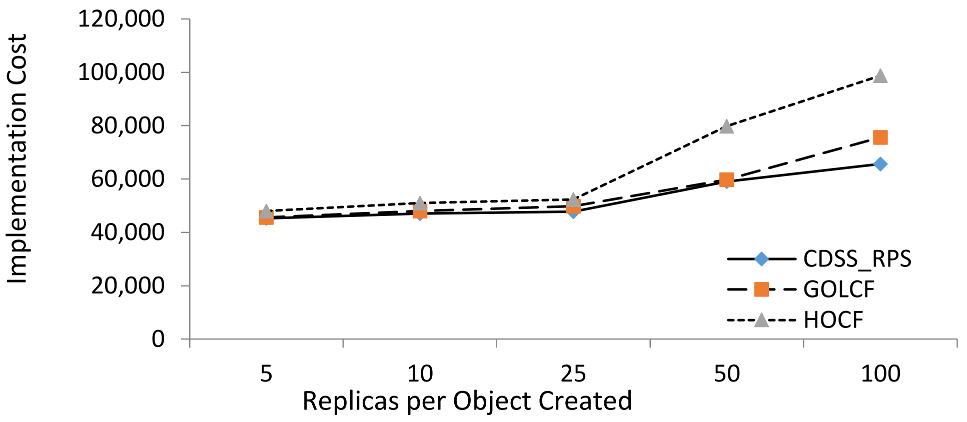


Figure 5. Schedule implementation cost w.r.t. CDSS_RPS, greedy object lowest cost first (GOLC) and highest opportunity cost first (HOCF).

As the amount of replicas per object increases, the probability of transfers and deletion also increases. Initially, the difference between the GOLCF and CDSS-RPS is very minor, because the probability of transfer action is low. With the increase of the number of replicas, the transfer action also increases, and this degrades the GOLCF and HOCF performances.

4.2.3. Geometric Mean of Turnaround Time

Figure 6 compares the geometric mean of turnaround time (GMTT) of CDSS-RPS, GOLCF, and HOCF. The number on the x-axis represents the number of jobs, and the y-axis represents the GMTT. A lower GMTT means an improved access time and performance from a user perspective. The GMTT does not allow applications or jobs that are too long. CDSS-RPS schedules the files and then replicates them on servers, then introduces an enhanced scheduling operator that handles the transfer and deletion action if required. This step-wise mechanism improves the GMTT, as shown in Figure 6 below.

The GMTT is calculated as:

$$\text{GMTT} = \left(\prod_{i=1}^{n} a_i\right)^{1/n} \tag{6}$$

where a is the turnaround time the job, i to n are the number of jobs in the concerned workload.

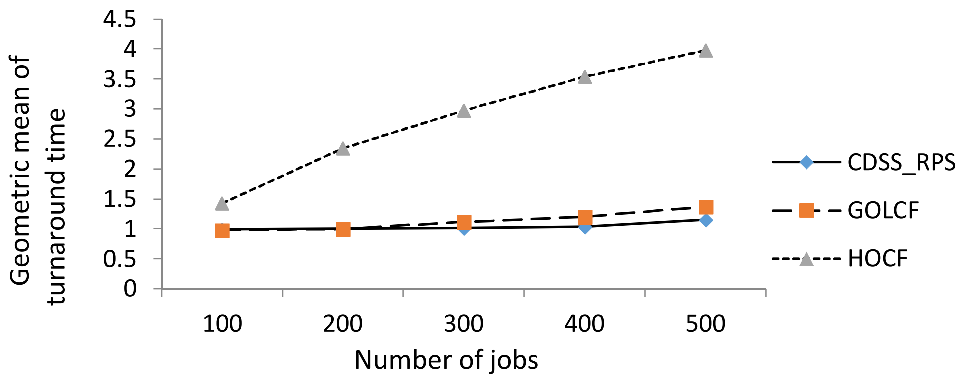


Figure 6. Geometric mean of turnaround time.

4.2.4. Execution Time

The experimental results of execution time for the number of jobs or files are shown in Figure 7. The execution time of a job is calculated as the sum of executing the job, which is the time used to transfer the file and process that job. Time reserved to transmit the file is the most important factor that impacts job execution time for jobs in a data grid. In Figure 7, the number on the x-axis represents the number of jobs, and the y-axis represents the execution time (in milliseconds). The execution time of CDSS-RPS is compared with that of GOLCF and HOCF.

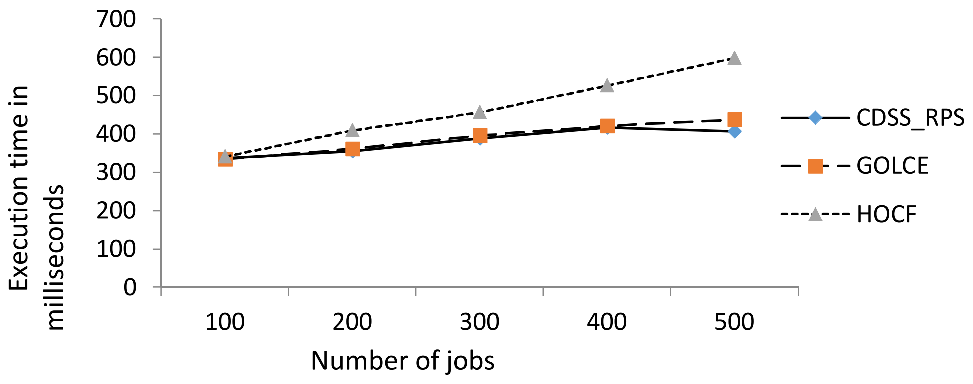


Figure 7. Execution time for all sets of jobs.

The turnaround time of 100 to 500 numbers of jobs is shown in a Figure 8 below, and the average turnaround time is 1.7.

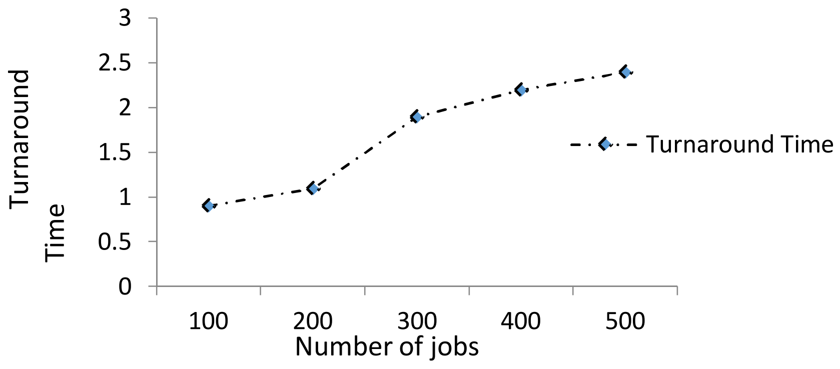


Figure 8. Turnaround time.

The above results show that CDSS-RPS works better than GOLCF and HOCF. GOLCF minimizes the implementation cost significantly, but does not schedule the files; this may increase the cost in the long run. CDSS-RPS schedules the files on its access basis; this minimizes the transfer and deletion probabilities, and ultimately improves the implementation cost. CDSS-RPS extends the work of [19] and gives better results than RTSP heuristics.

5. Conclusions and Future Work

In this paper, the centralized dynamic scheduling strategy-replica placement strategy (CDSS-RPS) is proposed. Results are formulated via simulations, also with reference to the plain RTSP algorithms. Our results indicate that the proposed CDSS-RPS algorithm provides an improved scheduling and replication mechanism compared to the RTSP heuristics. Though our paper shows significantly improved results, there is still a need for further testing of this strategy, in order to advance our knowledge in this area. The CDSS-RPS strategy can also be studied in terms of fault tolerance as well. For example, failure may occur in any computing sites, which may increase the implementation cost substantially. We have assumed a scenario where no failure occurs. It may also be worthwhile researching variants of the current problem. Investigating the faults and applying the fault-tolerant strategies may increase the execution time and the implementation cost as well, but the overall scheduling mechanism will be improved significantly. In the future, we plan to improve the replication strategy by further testing and comparisons with the latest techniques of a similar nature with regards to additional parameters.

Author Contributions: B.N. and F.I. designed the new techniques and performed the simulations and wrote the results. S.S. and A.T.C. contributed in directing the research and in writing the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Foster, I.; Kesselman, C. (Eds.) *The Grid 2: Blueprint for a New Computing Infrastructure*; Elsevier: New York, NY, USA, 2003.
2. Jolfaei, F.; Haghighat, A. Improvement of job scheduling and tow level data replication strategies in data grid. *Int. J. Mob. Netw. Commun. Telemat.* **2012**, *2*, 21–36. [CrossRef]
3. Balasangameshwara, J.; Raju, N. A hybrid policy for fault tolerant load balancing in grid computing environments. *J. Netw. Comput. Appl.* **2012**, *35*, 412–422. [CrossRef]
4. Abdi, S.; Pedram, H.; Mohamadi, S. The Impact of Data Replication on Job Scheduling Performance in Hierarchical data Grid. *arXiv*, 2010.
5. Berriman, G.B.; Deelman, E.; Good, J.C.; Jacob, J.C.; Katz, D.S.; Kesselman, C.; Laity, A.C.; Prince, T.A.; Singh, G.; Su, M.-H. Montage: A grid-enabled engine for delivering custom science-grade mosaics on demand. In Proceedings of the SPIE Astronomical Telescopes and Instrumentation, Glasgow, UK, 21–25 June 2004.
6. Mineter, M.J.; Jarvis, C.H.; Dowers, S. From stand-alone programs towards grid-aware services and components: A case study in agricultural modelling with interpolated climate data. *Environ. Model. Softw.* **2003**, *18*, 379–391. [CrossRef]
7. Foster, I.; Iamnitchi, A. On death, taxes, and the convergence of peer-to-peer and grid computing. In Proceedings of the International Workshop on Peer-to-Peer Systems, Berkeley, CA, USA, 21–22 February 2003.
8. Grace, R.K.; Manimegalai, R. Dynamic replica placement and selection strategies in data grids—A comprehensive survey. *J. Parallel Distrib. Comput.* **2014**, *74*, 2099–2108. [CrossRef]
9. Folling, A.; Grimme, C.; Lepping, J.; Papaspyrou, A. Robust load delegation in service grid environments. *IEEE Trans. Parallel Distrib. Syst.* **2010**, *21*, 1304–1316. [CrossRef]
10. Li, H. Realistic workload modeling and its performance impacts in large-scale escience grids. *IEEE Trans. Parallel Distrib. Syst.* **2010**, *21*, 480–493. [CrossRef]
11. Sonmez, O.; Mohamed, H.; Epema, D.H.J. On the benefit of processor coallocation in multicluster grid systems. *IEEE Trans. Parallel Distrib. Syst.* **2010**, *21*, 778–789. [CrossRef]

12. Amjad, T.; Sher, M.; Daud, A. A survey of dynamic replication strategies for improving data availability in data grids. *Future Gener. Comput. Syst.* **2012**, *28*, 337–349. [CrossRef]
13. Morris, J.H.; Satyanarayanan, M.; Conner, M.H.; Howard, J.H.; Rosenthal, D.S.; Smith, F.D. Andrew: A distributed personal computing environment. *Commun. ACM* **1986**, *29*, 184–201. [CrossRef]
14. Balman, M. Failure-Awareness and Dynamic Adaptation in Data Scheduling. Master's Thesis, Bogazici University, Istanbul, Turkey, 2008.
15. Tang, M.; Lee, B.; Tang, X.; Yeo, C.-K. The impact of data replication on job scheduling performance in the Data Grid. *Future Gener. Comput. Syst.* **2006**, *22*, 254–268. [CrossRef]
16. Mansouri, N.; Dastghaibyfard, G.H. Enhanced dynamic hierarchical replication and weighted scheduling strategy in data grid. *J. Parallel Distrib. Comput.* **2013**, *73*, 534–543. [CrossRef]
17. Bsoul, M.; Abdallah, A.E.; Almakadmeh, K.; Tahat, N. A round-based data replication strategy. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 31–39. [CrossRef]
18. Nicholson, C.; Cameron, D.G.; Doyle, A.T.; Millar, A.P.; Stockinger, K. Dynamic data replication in lcg 2008. *Concurr. Comput. Pract. Exp.* **2008**, *20*, 1259–1271. [CrossRef]
19. Loukopoulos, T.; Tziritas, N.; Lampsas, P.; Lalis, S. Implementing replica placements: Feasibility and cost minimization. In Proceedings of the Parallel and Distributed Processing Symposium, Long Beach, CA, USA, 26–30 March 2007.
20. GridSim. 2010. Available online: http://www.buyya.com/gridsim/ (accessed on 17 May 2018).
21. Horri, A.; Sepahvand, R.; Dastghaibyfard, G. A hierarchical scheduling and replication strategy. *Int. J. Comput. Sci. Netw. Secur.* **2008**, *8*, 30–35.
22. Ranganathan, K.; Foster, I. Simulation studies of computation and data scheduling algorithms for data grids. *J. Grid Comput.* **2003**, *1*, 53–62. [CrossRef]
23. Bsoul, M.; Al-Khasawneh, A.; Abdallah, E.E.; Kilani, Y. Enhanced fast spread replication strategy for data grid. *J. Netw. Comput. Appl.* **2011**, *34*, 575–580. [CrossRef]
24. Radoslavov, P.; Govindan, R.; Estrin, D. Topology-informed internet replica placement. *Comput. Commun.* **2002**, *25*, 384–392. [CrossRef]
25. Tang, X.; Xu, J. On replica placement for QoS-aware content distribution. In Proceedings of the INFOCOM 2004, Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, 6 February 2004; Volume 2.
26. Dowdy, L.W.; Foster, D.V. Comparative models of the file assignment problem. *ACM Comput. Surv.* **1982**, *14*, 287–313. [CrossRef]
27. Loukopoulos, T.; Lampsas, P.; Ahmad, I. Continuous replica placement schemes in distributed systems. In Proceedings of the 19th Annual International Conference on Supercomputing, Cambridge, MA, USA, 20–22 June 2005.
28. Maheswaran, M.; Ali, S.; Siegel, H.J.; Hensgen, D.; Freund, R.F. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *J. Parallel Distrib. Comput.* **1999**, *59*, 107–131. [CrossRef]
29. Hamscher, V.; Schwiegelshohn, U.; Streit, A.; Yahyapour, R. Evaluation of job-scheduling strategies for grid computing. In Proceedings of the Grid Computing—GRID, Bangalore, India, 17–20 December 2000; Springer: Berlin/Heidelberg, Germany, 2000; pp. 191–202.
30. Shan, H.; Oliker, L.; Biswas, R. Job superscheduler architecture and performance in computational grid environments. In Proceedings of the ACM/IEEE conference on Supercomputing, Phoenix, AZ, USA, 15–21 November 2003.
31. Park, S.-M.; Kim, J.H.; Ko, Y.B.; Yoon, W.S. Dynamic data grid replication strategy based on Internet hierarchy. In Proceedings of the International Conference on Grid and Cooperative Computing, Shanghai, China, 7 December 2003; Springer: Berlin/Heidelberg, Germany, 2004; pp. 838–846.
32. Chang, R.-S.; Chen, P.-H. Complete and fragmented replica selection and retrieval in Data Grids. *Future Gener. Comput. Syst.* **2007**, *23*, 536–546. [CrossRef]
33. Chang, R.-S.; Chang, J.-S.; Lin, S.-Y. Job scheduling and data replication on data grids. *Future Gener. Comput. Syst.* **2007**, *23*, 846–860. [CrossRef]
34. Abdi, S.; Mohamadi, S. The Impact of Data Replication on Job Scheduling Performance in Hierarchical Data Grid. *Int. J. Appl. Graph Theory Wirel. Ad Hoc Netw. Sens. Netw.* **2010**, *2*, 15–25. [CrossRef]
35. Abdi, S.; Mohamadi, S. Two level job scheduling and data replication in data grid. *Int. J. Grid Comput. Appl.* **2010**, *1*, 23–37. [CrossRef]

36. Sashi, K.; Thanamani, A.S. Dynamic replication in a data grid using a Modified BHR Region Based Algorithm. *Future Gener. Comput. Syst.* **2011**, *27*, 202–210. [CrossRef]
37. Andronikou, V.; Mamouras, K.; Tserpes, K.; Kyriazis, D.; Varvarigou, T. Dynamic QoS-aware data replication in grid environments based on data "importance". *Future Gener. Comput. Syst.* **2012**, *28*, 544–553. [CrossRef]
38. Taheri, J.; Lee, Y.C.; Zomaya, A.Y.; Siegel, H.J. A Bee Colony based optimization approach for simultaneous job scheduling and data replication in grid environments. *Comput. Oper. Res.* **2013**, *40*, 1564–1578. [CrossRef]
39. Vrbsky, S.V.; Galloway, M.; Carr, R.; Nori, R.; Grubic, D. Decreasing power consumption with energy efficient data aware strategies. *Future Gener. Comput. Syst.* **2013**, *29*, 1152–1163. [CrossRef]
40. Almuttairi, R.M.; Wankar, R.; Negi, A.; Rao, C.R.; Agarwal, A.; Buyya, R. A two phased service oriented Broker for replica selection in data grids. *Future Gener. Comput. Syst.* **2013**, *29*, 953–972. [CrossRef]
41. Gui, X.; Kui, Y. A Global Dynamic Scheduling with Replica Selection Algorithm Using GridFTP. In Proceedings of the 18th International Conference on Parallel and Distributed Computing and Systems, Bangkok, Thailand, 18–20 December 2006; Volume 1.
42. Li, W.; Yang, Y.; Yuan, D. Ensuring cloud data reliability with minimum replication by proactive replica checking. *IEEE Trans. Comput.* **2016**, *65*, 1494–1506. [CrossRef]
43. Souravlas, S.; Sifaleras, A. Binary-tree based estimation of file requests for efficient data replication. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 1839–1852. [CrossRef]
44. Liu, G.; Shen, H.; Chandler, H. Selective data replication for online social networks with distributed datacenters. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 2377–2393. [CrossRef]
45. Michelon, G.A.; Lima, L.A.; de Oliveira, J.A.; Calsavara, A.; de Andrade, G.E. A strategy for data replication in mobile ad hoc networks. In Proceedings of the IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), Banff, AB, Canada, 20–22 September 2017; pp. 486–489.
46. Nahir, A.; Orda, A.; Raz, D. Replication-based load balancing. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 494–507. [CrossRef]
47. Souli-Jbali, R.; Hidri, M.S.; Ayed, R.B. Dynamic data replication-driven model in data grids. In Proceedings of the IEEE 39th Annual Computer Software and Applications Conference (COMPSAC), Taichung, Taiwan, 1–5 July 2015; Volume 3, pp. 393–397.
48. Patil, R.; Zawar, M. Improving replication results through directory server data replication. In Proceedings of the International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli, India, 11–12 May 2017; pp. 677–681.
49. Khalajzadeh, H.; Yuan, D.; Grundy, J.; Yang, Y. Cost-Effective Social Network Data Placement and Replication using Graph-Partitioning. In Proceedings of the IEEE International Conference on Cognitive Computing (ICCC), Honolulu, HI, USA, 25–30 June 2017; pp. 64–71.
50. Usman, A.; Zhang, P.; Theel, O. A Component-Based Highly Available Data Replication Strategy Exploiting Operation Types and Hybrid Communication Mechanisms. In Proceedings of the IEEE International Conference on Services Computing (SCC), Honolulu, HI, USA, 25–30 June 2017; pp. 495–498.
51. Mansouri, N. A threshold-based dynamic data replication and parallel job scheduling strategy to enhance Data Grid. *Cluster Comput.* **2014**, *173*, 957–977. [CrossRef]
52. Zeng, L.; Veeravalli, B.; Zomaya, A.Y. An integrated task computation and data management scheduling strategy for workflow applications in cloud environments. *J. Netw. Comput. Appl.* **2015**, *50*, 39–48. [CrossRef]
53. Casas, I.; Taheri, J.; Ranjan, R.; Wang, L.; Zomaya, A.Y. A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems. *Future Gener. Comput. Syst.* **2017**, *74*, 168–178. [CrossRef]
54. Spaho, E.; Barolli, L.; Xhafa, F. Data replication strategies in P2P systems: A survey. In Proceedings of the 17th International Conference on Network-Based Information Systems (NBiS), Fiscianoon, Italy, 10–12 September 2014; pp. 302–309.
55. Souravlas, S.; Sifaleras, A. Trends in data replication strategies: A survey. *Int. J. Parallel Emerg. Distrib. Syst.* **2017**, 1–18. [CrossRef]

56. Loukopoulos, T.; Tziritas, N.; Lampsas, P.; Lalis, S. Investigating the Replica Transfer Scheduling Problem. In Proceedings of the 18th International Conference on Parallel and Distributed Computing and Systems (PDCS'06), Bangkok, Thailand, 1–18 August 2018.
57. Buyya, R.; Murshed, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurr. Comput. Pract. Exp.* **2002**, *14*, 1175–1220. [CrossRef]