

An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids

Srikumar Venugopal*, Rajkumar Buyya

Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, VIC 3010, Australia

Received 11 August 2006; received in revised form 30 March 2007; accepted 12 July 2007

Available online 20 July 2007

Abstract

Data-intensive Grid applications need access to large data sets that may each be replicated on different resources. Minimizing the overhead of transferring these data sets to the resources where the applications are executed requires that appropriate computational and data resources be selected. In this paper, we consider the problem of scheduling an application composed of a set of independent tasks, each of which requires multiple data sets that are each replicated on multiple resources. We break this problem into two parts: one, to match each task (or job) to one compute resource for executing the job and one storage resource each for accessing each data set required by the job and two, to assign the set of tasks to the selected resources. We model the first part as an instance of the well-known Set Covering Problem (SCP) and apply a known heuristic for SCP to match jobs to resources. The second part is tackled by extending existing MinMin and Sufferage algorithms to schedule the set of distributed data-intensive tasks. Through simulation, we experimentally compare the SCP-based matching heuristic to others in conjunction with the task scheduling algorithms and present the results.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Grid computing; Data-intensive applications; Task mapping

1. Introduction

Grids [16] aggregate computational, storage and network resources to provide pervasive access to their combined capabilities. In addition, Data Grids [12,19] provide services such as low latency transport protocols and data replication mechanisms to distributed data-intensive applications that need to access, process and transfer large data sets stored in distributed repositories. Such applications are commonly used by communities of researchers in domains such as high-energy physics, astronomy and biology.

The work in this paper is concerned with scheduling data-intensive applications that can be considered as a collection of independent tasks, each of which requires multiple data sets, onto a set of Grid resources. An astronomy image-processing

application following this model is described by Yamamoto et al. [46]. Each task is translated into a job that is scheduled on to a computational resource and requests data sets from the storage resources (or *data hosts*). Each of these data sets may be replicated at several locations that are connected to each other and to the computational sites (or *compute resources*) through networks of varying capability. This scenario is illustrated in Fig. 1. This paper makes two contributions: first, it introduces the problem of matching a task to a set of resources that consists of one compute resource for executing the job and a data host each to access each data set required for the job and models this problem as an instance of the well-known Set Covering Problem (SCP) and second, it applies a known heuristic for SCP to perform the matching and evaluates it against other strategies in conjunction with MinMin and Sufferage task scheduling algorithms through extensive simulations.

The rest of the paper is structured as follows: the next section presents a detailed resource model and the application model that is targeted in the research presented in this paper. The mapping heuristic is presented in the following section and is

* Corresponding author. Fax: +61 393481184.

E-mail addresses: srikumar@csse.unimelb.edu.au (S. Venugopal), raj@csse.unimelb.edu.au (R. Buyya).

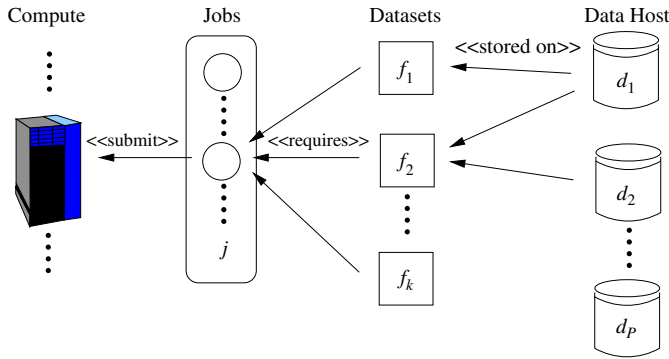


Fig. 1. Mapping problem.

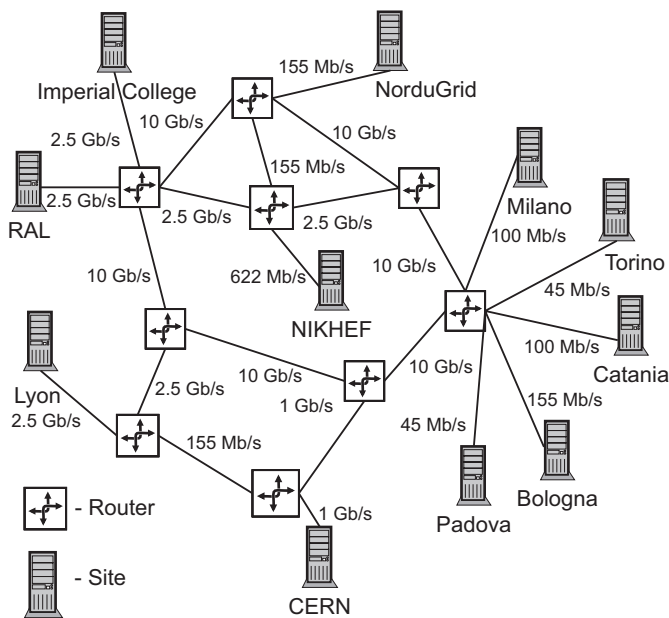


Fig. 2. European Data Grid Testbed 1 [3].

succeeded by details of experimental evaluation and the consequent results. Finally, the related work is presented and the paper is concluded.

2. Model

The target data-intensive computing environment is modelled based on existing production Grid testbeds such as the European DataGrid testbed [19] or the United States Grid3 testbed [17]. As an example, Fig. 2 shows a subset of European DataGrid Testbed 1 derived from Bell et al. [3]. The resources in the figure are spread across seven countries and belong to different autonomous administrative domains.

A data-intensive computing environment can be considered to consist of a set of M compute resources, $R = \{r_1, r_2, \dots, r_M\}$ and a set of P data hosts, $D = \{d_1, d_2, \dots, d_P\}$. Within production Grids, a compute resource is commonly a high performance computing platform such as a cluster consisting of processing nodes that are connected in a private local area

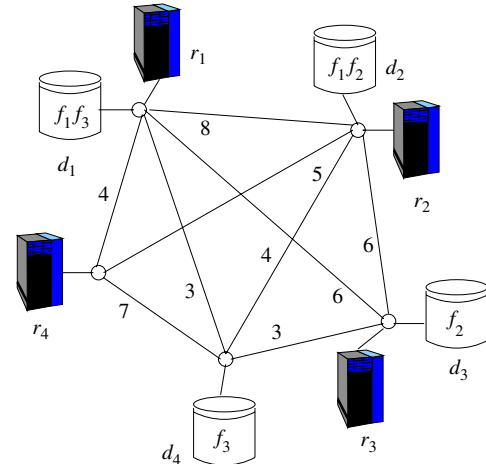


Fig. 3. A data-intensive environment.

network and are managed by a batch job submission system hosted at the “head” or “front-end” node connected to the public Internet. Jobs submitted to a cluster are assigned to processing nodes by the batch system or are held in queues. Such queues may have a limited capacity counted as the number of “slots” that can be filled by jobs. If all the slots in a queue are filled, further job submissions will not be allowed. A *data host* can be a dedicated storage resource such as a Mass Storage Facility connected to the Internet. At the very least, it may be a storage device attached to a compute resource in which case it inherits the network properties of the latter. It is important to note that even in the second case, the data host is considered separate from the compute resource. Fig. 3 shows a simplified data-intensive computing environment consisting of four compute resources and an equal number of data hosts connected by links of different bandwidths.

The physical network between the resources consists of entities such as routers, switches, links and hubs. However, the model presented here abstracts the physical network to consider the logical network topology wherein each compute resource is connected to every data host by a distinct network link as shown in Fig. 3. This logical link is denoted by $Link(r_m, d_p)$, $r_m \in R$, $d_p \in D$. The bandwidth of the logical link between two resources is the bottleneck bandwidth of the actual physical network between the resources and is given by $BW(Link(r_m, d_p))$. This information may be obtained from various information sources such as the Network Weather Service [45]. The numbers alongside the links in Fig. 3 depict the bandwidths of the various logical links in the network. The time taken by a compute resource to access data through the Internet is assumed to be an order of magnitude higher than that taken for it to access data on a storage resource at the same site (either a separate machine or a simple disk storage). Therefore, only remote access times are taken into account in the model and data sets on local storage have zero access times.

Data is organised in the form of data sets. A data set can be an aggregated set of files, a set of records or even a part of a large file. Data sets are replicated on the data hosts by a

separate replication process that follows a strategy such as one of those described by Bell et al. [3] which takes into consideration various factors such as locality of access, load on the data host and available storage space. Information about the data sets and their location is available through a catalog such as the Storage Resource Broker Metadata Catalog [33].

The application is composed of a set of N atomic (indivisible) and non-interdependent jobs, $J = \{j_1, j_2, \dots, j_N\}$ (Note: Since each task is translated into a job, tasks and jobs are used interchangeably). Typically, $N \gg M$, the number of compute resources. Each job, $j \in J$, requires a set of K data sets, denoted by F^j , that are distributed on a subset of D . Specifically, for a data set $f \in F^j$, $D_f \subseteq D$ is the set of data hosts (each denoted by d_f) on which f is replicated and from which it is available. Also, D_{f_1} and D_{f_2} need not be pairwise disjoint for every $f_1, f_2 \in F$. In other words, a data host can serve multiple data sets at a time.

Each job requires one processor in a compute resource for executing the job and one data host each for accessing each of the K data sets required by the job. The compute resource and the data hosts thus selected are collectively referred to as the *resource set* associated with the job and is denoted by $S^j = \{R^j, D^j\}$, where $R^j = \{r\}$, $r \in R$ represents the compute resource selected for executing the job and $D^j \subseteq \bigcup_{f \in F^j} D_f$ is the set of data hosts chosen for accessing the data sets required by the job.

The job execution time model followed here is extended from that presented by Maheswaran et al. [29]. Consider a job j that has been submitted for execution to a compute resource r . The time spent in waiting in the queue on the compute resource is denoted by $T_w(j, r)$ and the expected execution time of the job is given by $T_e(j, r)$. T_w increases with increasing load on the resource. Likewise, T_e is the time spent in purely computational operations and depends on the processing speed of the individual nodes within the compute resource. For each data set $f \in F^j$, the time required to transfer f from d_f to r is given by

$$T_t(f, d_f, r) = \text{Response_time}(d_f) \\ + \text{Size}(f)/\text{BW}(\text{Link}(d_f, r)),$$

$\text{Response_time}(d_f)$ is the difference between the time when the request was made to d_f and the time when the first byte of the data set f is received at r . This is an increasing function of the load on the data host. The *estimated completion time* for the job, $T_{ct}(j)$, is the wallclock time taken for the job from submission till eventual completion and is a function of these three times. Fig. 4 shows two examples of data-intensive jobs with times involved in various stages shown along a horizontal time-axis. In this figure, for convenience, the time for transferring f_1, f_2, \dots, f_k is denoted by $T_{f_1}, T_{f_2}, \dots, T_{f_k}$, respectively.

The impact of the transfer time of the data sets is dependent on the manner in which the data set is processed by the job. For example, Fig. 4(a) shows a common scenario in which Grid applications request and receive the required data sets in parallel before starting computation. In this case,

$$T_{ct}(j) = T_w(j, r) + \max_{f \in F^j} (T_t(f, d_f, r)) + T_e(j, r).$$

However, the number of simultaneous transfers on a link determines the bandwidth available for each transfer and therefore, the T_t .

Fig. 4(b) shows a more generic data processing approach in which some of the data sets are transferred completely prior to execution and the rest are accessed as streams during the execution. The grey areas show the overlap of computation and communication. In this case, the transfer time of the streamed data is masked by the computation time of the application. However, data access still affects the performance of the application. If there is a latency associated with accessing the data, the application may still have to wait until the first byte of the data is received at the compute resource.

This paper focuses on the application models of the first type, that is, applications that require all the data sets to be transferred to the actual compute resource (or its associated data host) before execution. This is the most common model followed by data-intensive applications [34]. Also, the impact of data transfer time is the highest in this model. However, it is possible that lessons learnt from scheduling these type of applications may also be applicable to the other types of data-intensive applications.

2.1. A generic scheduling algorithm

The scheduling paradigm followed is that of *offline* or *batch mode* scheduling of a set of independent tasks [29]. The general problem of creating a schedule for a set of jobs to run on distributed resources is called *list scheduling* and is considered to be *NP-complete* [5] in the general case. Many approximate heuristics have been devised for this problem and a short survey of these have been presented by Braun et al. [5]. Algorithm 1 outlines a generic strategy for batch mode scheduling of a set of jobs based on the skeleton presented by Casanova et al. [11].

Algorithm 1. A generic scheduling algorithm

```

1 while there exists unsubmitted jobs do
2   Update the resource performance data based on job scheduled
   in previous intervals
3   Update network data between resources based on current
   conditions
4   foreach unsubmitted job do
5     Match the job to a resource set to satisfy the objective
     function at the job level
6     Order the jobs depending on the overall objective
7   end
8   repeat
9     Assign mapped jobs to each compute resource heuristically
10  until all jobs are submitted or no more jobs can be submitted
11  Wait until the next scheduling event
12 end

```

The scheduler forms a part of a larger application execution framework such as a Grid resource broker (e.g. [37,44]). The resource broker is able to identify resources that meet minimum requirements of the application such as architecture (instruction set), operating system, storage threshold and data access permissions and these are provided as suitable candidates for job execution to the scheduler. The scheduling is carried out at

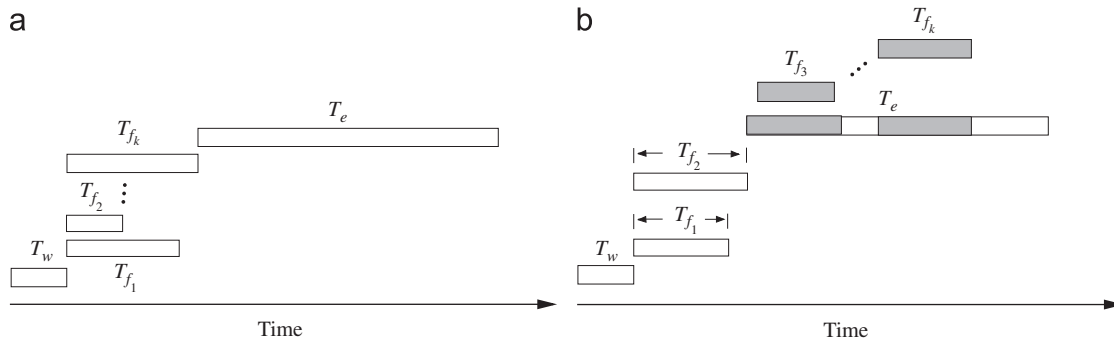


Fig. 4. Job execution stages and times (grey areas denote overlaps between the computation and data operations).

time intervals called *scheduling events* [28]. These events can be determined to either run at regular intervals (*poll-based*) or in response to certain conditions (*event-based*). There are two parts in a scheduling strategy: mapping and dispatching. The jobs have to be *matched* to a set of resources and ordered depending on the objective function (*mapping*) and then sent to remote resources for execution (*dispatching*). Each of the parts can be implemented independently and therefore, many strategies are possible.

The sections that follow concentrate on matching jobs to distributed resources where the selection of computational and data resources are interdependent. The aim of the matching heuristic is to select a resource set that produces the minimum completion time (MCT) for a job. The general strategy adopted here is to find a resource set with the least number of data hosts required to access the data sets required for a job and then, find a suitable compute resource to execute it. The goal here is to maximise the local access of data sets and thus, reduce the data transfer times.

3. A graph-based approach to the matching problem

For a job $j \in J$, consider a graph $G^j = (V, E)$ where $V = (\bigcup_{f \in F^j} \{D_f\}) \cup F^j$ and E is the set of all directed edges $\{d, f\}$ such that $d \in D_f$. Fig. 5(a) shows an example of a job j that requires 3 data sets f_1, f_2 and f_3 that are replicated on data host sets $\{d_1, d_2\}, \{d_2, d_3\}$ and $\{d_1, d_4\}$, respectively. The graph of data sets and data resources for job j is shown in Fig. 5(b).

The intuition followed in this work is to assign a job to a compute resource that is “closest” (in network terms) to a set of data hosts that contain the data sets required by the job. The selection of the compute resource, however, should not only be based on the proximity of the data but also on its availability and performance as well. In terms of the graph model presented, the resource set should therefore contain a set H of data hosts such that there exists atleast one edge from a member of H to f for every $f \in F^j$ in G^j so that all the data sets required for the job can be accessed. Fig. 5(c) shows a possible set for the graph of data sets and data hosts for job j shown in Fig. 5(b). There may be upto P^K possible sets of data hosts (as there may be upto P data hosts for each of the K data sets associated with a job) that can be combined with each compute resource

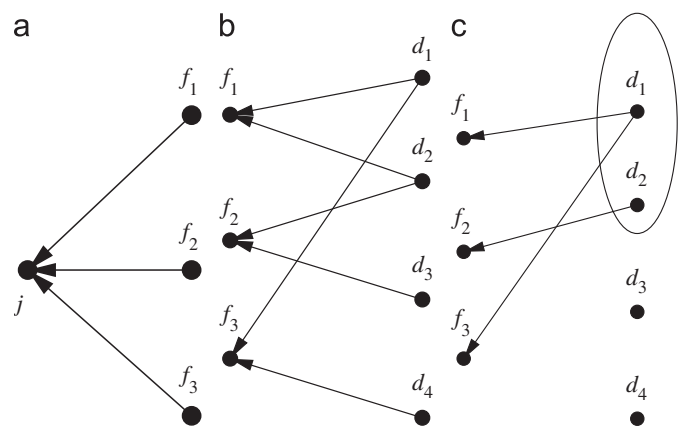


Fig. 5. Graph-based approach to the matching problem. (a) Job j dependent on 3 data sets. (b) Directed graph of data resources and data sets for job j . (c) A possible minimal set for the data graph.

in R to produce MP^K resource sets (where M is the number of compute resources) with different values of total completion time. Out of these, a combination of a set of data hosts and a compute resource is to be selected such that the total completion time for j is minimised. This problem is defined and referred to hereafter as the **Minimum Resource Set (MRS)** problem.

3.1. Modelling the MRS as a set cover

For a graph G^j such as that shown in Fig. 5(b), a reduced adjacency matrix $A = [a_{ik}], 1 \leq i \leq P, 1 \leq k \leq K$ can be constructed wherein $a_{ik} = 1$ if data host $d_i \in D_{f_k}$ for a data set f_k . Such an adjacency matrix is shown in Fig. 6(a). The rows that contain a 1 in a particular column are said to “cover” the column. The problem of finding the minimal set of data hosts to access all data sets in G^j is now equivalent to finding the set of the least number of rows such that every column is covered, that is, every column contains an entry of 1 in at least one of the rows. In other words, if each data host can be considered as a set of data sets, then finding the minimal set of data hosts is equivalent to finding the least number of such sets of data sets such that all data sets are covered. This problem has been studied extensively as the *SCP* [2].

$$\begin{array}{c}
 \text{a} \\
 \\
 \begin{array}{ccc}
 f_1 & f_2 & f_3 \\
 \begin{array}{c} d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}
 \end{array}
 \end{array}
 \quad
 \begin{array}{c}
 \text{b} \\
 \\
 \begin{array}{ccc}
 f_1 & f_2 & f_3 \\
 \begin{array}{c} d_1 \\ d_2 \\ d_3 \\ d_4 \end{array} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ - & - & - \\ d_1 & 1 & 0 & 1 \\ d_4 & 0 & 0 & 1 \end{pmatrix}
 \end{array}
 \end{array}
 \end{array}$$

Fig. 6. (a) Adjacency matrix for the job example, (b) tableau.

The SCP is an *NP-complete* problem and the most common approximation algorithm applied to the SCP is the Greedy strategy [14]. It is possible to derive a set cover for the data sets by following the Greedy strategy as outlined below:

Step 1: Repeat until all the data sets have been covered.

Step 2: \hookrightarrow Pick the data host that has the maximum number of uncovered data sets and add it to the current candidate set.

It can be seen that such a Greedy strategy will produce only one of the possible set covers. This, however, excludes the other candidate sets from consideration. It is also possible to arrive at an exhaustive search procedure that generates all the possible covers. However, this is bound to be computationally intensive.

The next subsection details a heuristic for matching jobs to resources based on the approximate Tree Search algorithm provided by Christofides [13] for the SCP. This heuristic restricts the search to a region where the solution is most likely to be found. But, before applying that algorithm, it is possible to reduce the size of the problem by taking advantage of the nature of the SCP. These reductions are as follows:

- (1) If a data set required for a job is present on only one data host, then that data host is part of any solution. Therefore, the problem can be reduced by assigning the data set to that data host and removing the data set from later consideration.
- (2) For $f_1, f_2 \in F^j$, if $D_{f_1} \subseteq D_{f_2}$, then f_2 can be removed from consideration as any solution that covers f_1 must also cover f_2 .

3.2. The SCP Tree Search heuristic

Algorithm 2 outlines the SCP Tree Search heuristic that consists of three distinct phases: initialisation, execution and termination. These are described in the following paragraphs. Fig. 7 shows an example of the heuristic in action based on the job shown in Fig. 5 using the tableau in Fig. 6(b) and the platform in Fig. 3. At the bottom of each step, the candidate resource set arrived at is depicted as well.

Initialisation (lines 1–3): The initialisation starts off with the creation of the adjacency matrix A for a job. The rows of this matrix (that is, the data hosts) are then sorted in the descending order of number of 1's per column (or, the number of data sets contained). This sorted matrix is used to create an augmented matrix that is henceforth referred to as the *tableau* and is shown

Algorithm 2. SCP Tree Search matching heuristic

Begin Main

- 1 For a job j , create the adjacency matrix A with data hosts forming the rows and data sets forming the columns
 - 2 Sort the rows of A in the descending order of the number of 1's in a row
 - 3 Create the tableau T from sorted A and begin with initial solution set $B_{\text{final}} = \phi$, $B = \phi$, $E = \phi$ and $z = \infty$
 - 4 Search(B_{final} , B , T , E , z)
 - 5 $S^j \leftarrow \{r\}$, B_{final} where $r \in R$ such that S^j produces MCT(B_{final})
- End Main*

Search(B_{final} , B , T , E , z)

- 6 Find the minimum k , such that $f_k \notin E$. Let T_k be the block of rows in T corresponding to f_k . Set a pointer q to the top of T_k .
 - 7 **while** q does not reach the end of T_k **do**
 - 8 $F_T \leftarrow \{f_i | t_{qi} = 1, 1 \leq i \leq K\}$
 - 9 $B \leftarrow B \cup \{d_q^k\}$, $E \leftarrow E \cup F_T$
 - 10 **if** $E = F^j$ **then**
 - 11 **if** $z > \text{MCT}(B)$ **then**
 - 12 $B_{\text{final}} \leftarrow B$, $z \leftarrow \text{MCT}(B)$
 - 13 **else** Search(B_{final} , B , T , E , z)
 - 14 $B \leftarrow B - \{d_q^k\}$, $E \leftarrow E - F_T$
 - 15 Increment q
 - 16 **end**
- MCT(B)
- 17 Find $r \in R$ such that the completion time is minimum for the resource set $S^j = \{r, B\}$ and return value
-

in Fig. 6(b). The tableau T consists of K blocks of rows (delimited by dashes in Fig. 6(b)), where K is the size of F^j and the k th ($1 \leq k \leq K$) block consists of rows corresponding to data hosts that contain f_k , $f_k \in F^j$. The tableau is constructed in such a manner that the rows within each block are in the same sorted order as the rows in the sorted adjacency matrix. At any stage of execution, the set of data hosts B keeps track of the current solution set of datahosts, the set E contains the data sets already covered by the solution set and the variable z keeps track of the value of the completion time offered by the current solution set. The final solution set is stored in B_{final} . The procedure begins with the partial solution set $B = \phi$, $E = \phi$, $z = \infty$.

Execution (lines 6–16): During execution, the blocks are searched sequentially starting from the k th block in T where k is the smallest index, $1 \leq k \leq K$ such that $f_k \notin E$. Within the k th block, let d_q^k mark the data host under consideration where q is a row pointer within block k . The data host d_q^k is added to B and all the data sets for which the corresponding row contains 1 are added to E as they are already covered by d_q^k . These data sets are removed from consideration and the process then moves to the next uncovered block until $E = F^j$, that is, all the data sets have been covered. At this point, B represents the corresponding candidate set of data hosts that covers all the data sets. The function MCT(B) computes the expected value of the completion time for each compute resource combined with B and returns with the minimum of the values so found. If this is lower than the existing value in z , then the solution set is replaced with the current candidate set and z is assigned the returned value.

a	b	c
f_1 f_2 f_3	f_1 f_2 f_3	f_1 f_2 f_3
d_1 1 0 1	d_1 1 0 1	d_1 1 0 1
d_2 1 1 0	d_2 1 1 0	d_2 1 1 0
— — —	— — —	— — —
d_2 1 1 0	d_2 1 1 0	d_2 1 1 0
d_3 0 1 0	d_3 0 1 0	d_3 0 1 0
— — —	— — —	— — —
d_1 1 0 1	d_1 1 0 1	d_1 1 0 1
d_4 0 0 1	d_4 0 0 1	d_4 0 0 1
	$\{\{r_1\}, d_1, d_2\}$	$\{\{r_1\}, d_1, d_3\}$
d	e	f
f_1 f_2 f_3	f_1 f_2 f_3	f_1 f_2 f_3
d_1 1 0 1	d_1 1 0 1	d_1 1 0 1
d_2 1 1 0	d_2 1 1 0	d_2 1 1 0
— — —	— — —	— — —
d_2 1 1 0	d_2 1 1 0	d_2 1 1 0
d_3 0 1 0	d_3 0 1 0	d_3 0 1 0
— — —	— — —	— — —
d_1 1 0 1	d_1 1 0 1	d_1 1 0 1
d_4 0 0 1	d_4 0 0 1	d_4 0 0 1
	$\{\{r_2\}, d_2, d_1\}$	$\{\{r_2\}, d_2, d_4\}$

Fig. 7. Example of the SCP Tree Search heuristic in action.

Whenever the heuristic enters a block that is not yet covered, it branches out within the block by a recursive call that passes along the incomplete solution set (line 13). The final solution set is returned in the variable B_{final} through normal pass-by-reference methods. At the end of each loop, the heuristic backtracks to try the next data host in the block and repeat the branching with that host (line 14).

Illustrating this using Fig. 7, in the first step (Fig. 7(a)), the heuristic starts with the first block in the tableau. As f_1 and f_3 are covered by choosing d_1 , the heuristic moves to the second block to cover f_2 . This gives us $B = \{d_1, d_2\}$ to cover all the data sets. For this B , r_1 produces the lowest value of MCT and therefore, the candidate resource set (or the resource set matched to the job) at this moment is $\{\{r_1\}, d_1, d_2\}$ (Fig. 7(b)). The heuristic then backtracks and moves to the next row in the same block to produce $B = \{d_1, d_3\}$ and the consequent candidate resource set $\{\{r_1\}, d_1, d_2\}$ (Fig. 7(c)). The latter is then compared to the previous resource set and the B with lowest MCT is selected for the next iteration. In a similar fashion (Figs. 7(d)–(f)), the rest of the resource sets are discovered and a final resource set selected.

Termination (line 5): Through the recursive procedure outlined in the listing, the heuristic then backtracks and discovers other candidate sets. The solution set that guarantees minimum makespan is then chosen as the final solution set. The compute resource that provides the MCT is then combined with the solution set to obtain the resource set for the job.

To reduce the scope of the tree traversal, the heuristic terminates when the first block is exhausted. The data hosts with the maximum number of data sets appear at the top of the tableau due to the initialisation process. Therefore, most of the candidate sets will be covered by the search function by starting at the rows in the first block.

4. Other approaches to the matching problem

Algorithm 3. The Compute-First matching heuristic

```

1. foreach  $j \in J$  do
2.   Let  $S^j \leftarrow \{R^j, D^j\}$ ,  $R^j \leftarrow \phi$ ,  $D^j \leftarrow \phi$ 
3.   Let  $R^j \leftarrow \{r_{\text{final}}\}$  such that  $T_e(j, r_{\text{final}})$  is minimum for all
      $r \in R$ 
4.   foreach  $f \in F^j$  do
5.      $D^j \leftarrow D^j \cup \{d_f\}$  where  $T_i(f, d_f, r_{\text{final}})$  is minimum for
       all  $d_f \in D_f$ 
6.   end
7. end

```

Compute-First—In this mapping strategy, listed in Algorithm 3, the compute resource that provides the least execution time is selected first. This step is followed by choosing data hosts that have the highest bandwidths (and therefore, the lowest transfer times) to the selected compute resource. The running time of this heuristic is $O(MKP)$.

Algorithm 4. The Exhaustive Search matching heuristic

```

1. foreach  $j \in J$  do
2.   Let  $S^j \leftarrow \{R^j, D^j\}$ ,  $R^j \leftarrow \phi$ ,  $D^j \leftarrow \phi$ 
3.   Let  $U \leftarrow R \times D_{f_1} \times D_{f_2} \times \dots \times D_{f_K}$  where
      $f_1, f_2, \dots, f_K \in F^j$ 
4.   Find  $u \in U$  such that  $T_{\text{ct}}(j)$  is minimum
5. end

```

Exhaustive Search—Algorithm 4 lists the exhaustive search strategy wherein all the possible resource sets for a particular job are generated and the one guaranteeing the least completion time is chosen for the job. While this heuristic guarantees that the resource set selected will be the best for the job, it searches through MP^K resource sets at a time. This leads to unreasonably large search spaces for higher values of K . For example, for a job requiring 5 data sets with 20 possible data hosts and 20 available compute resources, the search space will consist of $(20 * 20^5) = 64 * 10^6$ resource sets.

Greedy Selection—This strategy, listed in Algorithm 5, builds the resource set by iterating through the list of data sets and making a Greedy choice for the data host for accessing each data set, followed by choosing the best compute resource for that data host. At the end of each iteration, it checks whether the compute resource so selected is better than the

one selected in previous iteration when the data hosts selected in previous iterations are considered.

Algorithm 5. Greedy Selection strategy

```

1 foreach  $j \in J$  do
2   Let  $S^j \leftarrow \{R^j, D^j\}$ ,  $R^j \leftarrow \phi$ ,  $D^j \leftarrow \phi$ 
3   Let  $R_{temp}^j \leftarrow \phi$  // A temporary variable
4   foreach  $f \in F^j$  do
5     Let  $U \leftarrow \{(d_f, r)\}_{d_f \in D_f}$  where  $r$  is the first element of
       ordered set  $R_{d_f}$ 
6     Find  $(d_f, r)$  such that  $T_t(f, d_f, r) + T_e(j, r)$  is minimum
       over  $U$ 
7     if  $S^j = \{\phi, \phi\}$  then
8        $R^j \leftarrow \{r\}$ ,  $D^j \leftarrow \{d_f\}$ ,  $R_{temp}^j \leftarrow \{r\}$ 
9     else
10       $R^j \leftarrow \{r\}$ ,  $D^j \cup \{d_f\}$ 
11       $S^j \leftarrow \min\{\{R^j, D^j\}, \{R_{temp}^j, D^j\}\}$ 
12       $R_{temp}^j \leftarrow R^j$ 
13   end
14 end

```

This heuristic was presented by the authors [43] for deadline and budget constrained cost and time minimisation scheduling of distributed data-intensive applications. The running time of this heuristic is $O(MKP)$.

5. Scheduling heuristics

While the mapping heuristic finds a resource set for a single job, the overall objective is to minimise the total *makespan* [29], the total time from the start of the scheduling to the completion of the last job, of the application consisting of N such data-intensive jobs. To that end, we apply the well-known MinMin and Sufferage heuristics, proposed by Maheswaran et al. [29], for dynamic scheduling of jobs on heterogeneous computing resources. These have been extended to take into account the distributed data requirements of the target application model.

Algorithm 6 outlines the extended MinMin scheduling heuristic. The basic idea of this heuristic is to find the job that has the least value of completion time among all the jobs and allocate it to the resource set that achieves it. The intuition behind this is that such an allocation over all the jobs will minimise the overall completion time. The term J_U denotes the set of jobs that have not been allocated to any resource set yet. In the beginning, it matches all the jobs to a resource set that guarantees the MCT for that job (line 4). This is produced through matching heuristics such as the SCP Tree Search, Greedy Selection, Compute-First or Exhaustive Search, that have been presented in previous sections. Then, the job with the MCT in the present allocation is assigned to the compute resource in its chosen resource set (line 7). This job is then removed from the unallocated job set. As job assignment changes the availability of the compute resource with respect to the number of available slots/processors, the resource information is updated and the process is repeated until all the jobs in J_U have been allocated to some resource set.

Algorithm 6. The MinMin Scheduling heuristic extended for distributed data-intensive applications

```

1 repeat
   Begin Mapping
2   repeat
3     foreach  $j \in J_U$  do
4       Find the resource set that achieves the MCT for  $j$ 
5     end
6     Find the job  $j \in J_U$  with the least value of  $T_{ct}(j)$ 
7     Assign  $j$  to its selected resource set and remove it from  $J_U$ 
8     Update the resource availability based on the allocation
       performed in the previous step
9   until  $J_U$  is empty
   End Mapping
10  Dispatch the mapped jobs to the selected resources such that
     the job allocation limit of each resource is not exceeded
11  Wait until the next scheduling event
12  foreach job completed in the previous interval do
13    For each data set that has been transferred from a remote
       data host for the job, add its eventual destination
       (compute resource) as a future source of the data set for
       the jobs remaining in  $J_U$ 
14  end
15  For each resource, revise its capability estimates (job
     allocation limit or available queue slots) depending on various
     information sources such as external performance monitors or
     the jobs completed in the previous interval
16 until all jobs are completed

```

Algorithm 7. The Sufferage heuristic extended for distributed data-intensive applications

```

   Begin Mapping
1 repeat
2   foreach  $j \in J_U$  do
3     Find the resource set that achieves the MCT for  $j$ 
4     Find the second best completion time for  $j$ 
5     sufferage value = second best value – best value
6   end
7   Find the job  $j \in J_U$  with the maximum sufferage value
8   Assign  $j$  to its selected resource set and remove  $j$  from  $J_U$ 
9   Update the resource availability based on the allocation
     performed in the previous step
10 until  $J_U$  is empty
   End Mapping

```

For each compute resource, the dispatching function (line 10) submits the jobs mapped to it to the remote job management system (or the job queue) until all the slots on the queue have been filled or the jobs exhausted. The remaining jobs that were assigned to the compute resource but were not able to be allocated to the remote queues, are returned back to the unallocated jobs list. The scheduler then waits until the next scheduling event to resume.

When a job is scheduled for execution on a compute resource, all the data sets that are required for the job and are not available local to the resource, are transferred to the resource prior to execution. These data sets become replicas that can be used by following jobs. Here, this is taken into account by registering the compute resource in question (or its associated data host) as a source of the transferred data sets for succeeding allocation loops (line 13). This enables exploiting both temporal and spatial locality of data access.

The motivation behind the Sufferage heuristic (listed in Fig. 7) is to allocate a resource set to a job that would be disadvantaged the most (or “suffer” the most) if that resource set were not allocated to it. This is determined through a sufferage value computed as the difference between the second best and the best value of the completion time for the job.

For each job, the resource set that offers the least value of the completion time is determined through the same mechanisms as that in MinMin. Then, the compute resource in that resource set is removed from consideration and the matching function is rerun to provide another minimal resource set with the next best value for the completion time. The selection of the compute resource determines both the execution time (T_e) and the data transfer times (T_t). Therefore, removing it from consideration will produce the maximum impact on the value of the completion time. After determining the sufferage value for each job, the job with the largest sufferage value is then selected and assigned to its chosen resource set. The rest of the heuristic including dispatching and updating of compute resource and data host information proceeds in the same manner as MinMin.

6. Evaluation of scheduling algorithms

Effective evaluation of scheduling algorithms requires the study of their performance under different scenarios such as different user inputs and varying resource conditions. Within Grid environments, resource loads and the number of users vary continuously and the spread of resources among different administrative domains makes it nearly impossible to control the environment to provide a stable configuration for evaluation. Furthermore, the network plays a large role in the performance of scheduling algorithms for data-intensive applications and it is impossible to create consistent conditions over public networks. The scale of the evaluation is also limited by the number of Grid resources that can be accessed.

Therefore, it was decided to evaluate the performance of algorithms on a simulated Grid environment to ensure a stable and repeatable configuration. Simulation has been used extensively for modelling and evaluation of distributed computing systems and the popularity of this methodology for evaluation of Grid scheduling algorithms have led to the availability of several Grid simulation packages [41]. Some of the simulation systems available for data-intensive computing environments such as Data Grids include GridSim [8], MONARC simulator [27], OptorSim [3], ChicSim [35] and SimGrid [10]. GridSim enables modelling and simulation of heterogeneous Grid resources with time-shared and space-shared node allocation and different economic costs; Grid networks with different routing topologies and QoS classes [40]; and Data Grid replica catalogs that can be connected in different configurations [39]. Also, it presents itself as a toolkit that allows creation of different applications such as resource brokers having scheduling algorithms with different objectives. Hence, GridSim was used as the simulation system for evaluating the scheduling algorithms for distributed data-intensive applications. Evaluation of the scheduling algorithms in GridSim required modelling of Grid resources, their interconnections and the data-intensive

applications. The sections that follow describe in detail how each of these were modelled.

6.1. Simulated resources

The testbed modelled in this evaluation is shown in Fig. 2. The modelled testbed contains 11 resources spread across six countries connected via high capacity network links. Each resource, except the one at CERN (Geneva), was used both as a compute resource and as a data host. The resource at CERN was used as a pure data source (data host) in the evaluation and therefore, no jobs were submitted to it for execution. The resources in the actual testbed have gone through several configuration changes, not all of which are publicly available, and hence it was impossible to model their layout and CPU capability accurately. Instead, it was decided to create a configuration for each resource such that the modelled testbed, in whole, would reflect the heterogeneity of platforms and capabilities that is normally the characteristic of Grids. All the resources were simulated as clusters of single CPU nodes or Processing Elements (PEs) with a batch job management system using space-shared policy. This modelled real world Grid resources that are generally high performance clusters in which each job is allocated to a processing node through a job submission queue. The processing capabilities of the PEs were rated in terms of Million Instructions Per Sec (MIPS) so that the application requirements can be modelled in Million Instructions (MI). The configuration assigned to the resources in the testbed for the simulation are listed in Table 1.

To model resource contention caused by multiple users submitting jobs simultaneously and the resultant variation in resource availability, a *load factor* was associated with each resource. The load factor is simply the ratio of the number of PEs that are occupied to the total number of PEs available in a resource. During simulation, the instantaneous load (or number of PEs occupied) for each resource was derived from a Gaussian distribution centred around its mean load factor shown in Table 1.

Storage at the resources was modelled as the total disk capacity available at the site. Site access latencies such as disk read time were ignored as these are less than the network delays by an order of magnitude. The network between the resources were modelled as the set of routers and links shown in Fig. 2. Variations of the available network bandwidth are simulated by associating a link load factor, which is the ratio of the available bandwidth to the total bandwidth for a network link. During simulation, the instantaneous measure of the link load is derived from another Gaussian distribution centred around a mean load assigned at random, at the start of the simulation, to each of the links.

It was possible to keep track of the various load variations through information services built into the simulation entities. For example, it was possible to query the instantaneous bandwidth of the network link between any two resources. It was also possible to determine resource availability information by querying the resource for its instantaneous load and number of PEs available.

Table 1
Resources within EDG testbed used for evaluation

Resource name (location)	No. of nodes	Single PE rating (MIPS)	Storage (TB)	Mean load
RAL (UK)	41	1140	2.75	0.9
Imperial College (UK)	52	1330	1.80	0.95
NorduGrid (Norway)	17	1176	1.00	0.9
NIKHEF (Netherlands)	18	1166	0.50	0.9
Lyon (France)	12	1320	1.35	0.8
CERN (Switzerland)	–	–	12	–
Milano (Italy)	7	1000	0.35	0.5
Torino (Italy)	4	1330	0.10	0.5
Catania (Italy)	5	1200	0.25	0.6
Padova (Italy)	13	1000	0.05	0.4
Bologna (Italy)	20	1140	5.00	0.8

6.2. Distribution of data

A universal set of 1000 data sets was used for this evaluation. Studies of similar environments [31] have shown that the size of the data sets follow a heavy-tailed distribution in which there are larger numbers of smaller size data sets and vice versa. Therefore, the set of data sets are generated with sizes distributed according to the logarithmic distribution in the interval [1 GB, 6 GB]. The distribution of data sets in a Data Grid depends on many factors including variations in popularity, the replication strategy employed and the nature of the Grid fabric. To model this distribution, at the start of the simulation, each of the data sets were replicated on one or more of the data hosts according to a preset pattern of data set distribution. Two common patterns of data distribution considered in this evaluation are given below:

- *Uniform*: Here, the distribution of data sets is modelled on a uniform distribution. Here, each data set is equally likely to be replicated at any site.
- *Zipf*: Zipf-like distributions follow a power law model in which the probability of occurrence of the i th ranked data set in a list of data sets is inversely proportional to i^{-a} where $a \leq 1$. In other words, a few data sets are distributed widely whereas most of data sets are found in one or two places. This models a scenario where the data sets are replicated on the basis of popularity. It has been shown that Zipf-like distributions holds true in cases such as requests for pages in World Wide Web where a few of the sites are visited the most [6]. This scenario has been evaluated for a Data Grid environment in related publications [9].

Henceforth, the distribution applied is described by the variable *Dist*. The distribution of data sets was also controlled through a parameter called the *degree of replication* which is the maximum possible number of replicas of any data set present in the Data Grid at the beginning of the simulation. For example, a degree of replication of 3 means there can be up to 3 copies of any data set on the Grid resources. However, not all data sets are replicated to the limit of the degree of replication. In a uniform distribution, a higher percentage of the data sets are replicated up to the maximum limit than in the Zipf distribution. The degree of replication in this evaluation is 5.

6.3. Application and jobs

The simulated application models a Bag-of-Task application that can be converted into a set of independent jobs. The size of the application was determined by the number of jobs in the set (or N). Each job translates to a Gridlet object which is the smallest unit of execution in GridSim. The computational size of a job or the job length, described by the term *Size*, is expressed in terms of the time taken to run the job on a standard PE with a MIPS rating of 1000. That is, a job with length 100,000 MI runs for 100 s on a standard resource. Each job requires as input, a pre-determined number of data sets (or K data sets) selected at random from the universal set of data sets. For the purpose of comparison, K is kept a constant among all the jobs in a set although this is not a condition imposed on the heuristic itself.

An experiment is an execution of the all the heuristics for an application while keeping the values for these parameters constant, and is therefore described by the tuple $(N, K, Size, Dist)$. At the beginning of each experiment, the set of data sets, their distribution among the resources, and the set of jobs are generated. This configuration is then kept constant while each of the scheduling heuristics are evaluated in turn. To keep the resource and network conditions repeatable among evaluations, a random number generator is used with a constant seed. The evaluation is conducted with different values for N , K , *Size* and *Dist* to study the performance under different input conditions.

7. Experimental results

7.1. Comparison between the matching heuristics

The performances of the matching heuristics discussed in the previous section were compared with each other by pairing each of them with the MinMin heuristic and conducting 50 simulation experiments with different values for N , K , *Size* and *Dist*. Throughout this section, *SCP* and *Greedy* refer to the SCP Tree Search and the Greedy Selection heuristics presented in the previous section respectively. The objective of this evaluation was to reduce the *makespan* [29] of the application which is the total wallclock time between the submission of the first job to the completion of the last job in the set.

Table 2
Summary of simulation results

Mapping heuristic	Geometric mean	Avg. deg. (SD)	Avg. rank (SD)
Compute-First	37,593.71	69.01 (19.4)	3.63 (0.48)
Greedy	36,927.44	71.86 (50.55)	3.23 (0.71)
SCP	24,011.17	7.68 (10.42)	1.67 (0.6)
Exhaustive search	23,218.49	3.87 (6.46)	1.47 (0.58)

The results of the experiments are summarised in Table 2 and are based on the methodology provided by Casanova et al. [11]. For each matching heuristic, the table contains three values:

- (1) *Geometric mean* of the makespans: The geometric mean is used as the makespans vary in orders of magnitude depending on parameters such as number of jobs per application set, number of files per job and the size of each job. The lower the geometric mean, the better the performance of the heuristic.
- (2) *Average degradation (Avg. deg.)* from the best heuristic: In an experiment, the degradation of a heuristic is the difference between its makespan and the makespan of the best heuristic for that experiment and is expressed as a percentage of the latter measure. The average degradation is computed as an arithmetic mean over all experiments and the standard deviation of the population is given in the parentheses next to the means in the table. This is a measure of how far a heuristic is away from the best heuristic for an experiment. A lower number for a heuristic certainly means that on an average that heuristic is better than the others.
- (3) *Average rank (Avg. rank)* of each heuristic in an experiment: The ranking is in the ascending order of makespans produced by the heuristics for each experiment, that is, the lower the makespan, the lower the rank of the heuristic. The average rank is calculated over all the experiments and the standard deviation is provided alongside the averages in parentheses.

The three values together provide a consolidated view of the performance of each heuristic. For example, it can be seen that on average Compute-First and Greedy both perform worse than either SCP or Exhaustive Search. However, the standard deviation of the population is much higher in the case of Greedy than that of Compute-First. Therefore, Compute-First can be expected to perform as the worst heuristic most of the time. Indeed, in a few of the experiments, Greedy performed as good or even better than SCP while Compute-First never came close to the performance of the other heuristics.

As is expected, between SCP and Exhaustive Search, the latter provides the better results by having a consistently lower score than the former. However, the nature of Exhaustive Search means that as the number of data sets per job increases, the number of resource sets that need to be considered by the heuristic increases dramatically. The geometric mean and average rank of SCP is close to that of Exhaustive Search heuristic. The average rank is less than 2 for both heuristics which implies that

in many scenarios, SCP provides a better performance than Exhaustive Search.

7.1.1. Impact of data transfer on performance

Figs. 8–10 show a more fine-grained view of the experimental evaluation by showing the effect of varying one of the variables (N , K , $Size$, $Dist$), all others kept constant. Essentially, these are snapshots of the experimental results that contributed to the summary data in Table 2. Along with the makespan, two more measures of performance are considered within these figures. These are:

- (1) *Mean percentage of data time*: For each job in an experiment, the share of the data transfer time is calculated as a percentage of the total execution time for that job. The average of this measure over all the jobs then represents the mean impact of the data transfer time on the set of jobs or the application as a whole. A lower number is better as one of the aims of the scheduling algorithms presented so far has been to reduce the data transfer time.
- (2) *Mean locality of access*: For each job, the ratio of the number of data sets accessed from the local disk storage of the compute resource to the total number of data sets accessed by the job from all resources is calculated as a percentage of the latter and is termed as the *local access ratio*. The average of the local access ratio over all the jobs becomes a measure of locality exploited by each of the algorithms. In this case, a higher number is better as increased local access decreases the impact of remote data transfer on the performance.

These two measures represent two slightly different perspectives on the data access performed by the jobs. Consider a job that requires one data set of size 6 GB and two data sets of size 1 GB each. The job may be scheduled such that the larger-sized data set is accessed locally, whereas the smaller-sized data sets may be accessed from remote data hosts. In this case, the data transfer component is small but the locality of access is low as well. However, when the sizes of the data sets are more or less equal, the locality of access becomes an important factor. These two measures, therefore, give an indication of the importance given by the algorithms to the location of data. These can be correlated with the makespan to judge the impact of the selection made by an algorithm on its performance.

Fig. 8 shows the impact of the number of jobs on the performance of the algorithm. It can be seen that as the number of jobs increases, the makespan of Compute-First and Greedy heuristic rise more steeply than the other two. The impact of data time is lower for SCP and Exhaustive Search than it is for Compute First and is a factor in their improved performance. Locality of access is also higher for the former two algorithms and it increases as the number of jobs in the set increases. This is because the probability of data sets being shared increases with more jobs accessing the same global set of data sets as was the case in this evaluation. This means that there is a greater chance for transferred data sets to be reused with a higher number of jobs. In case of Zipf distribution (right column), the locality is lower than in the case of Uniform distribution which

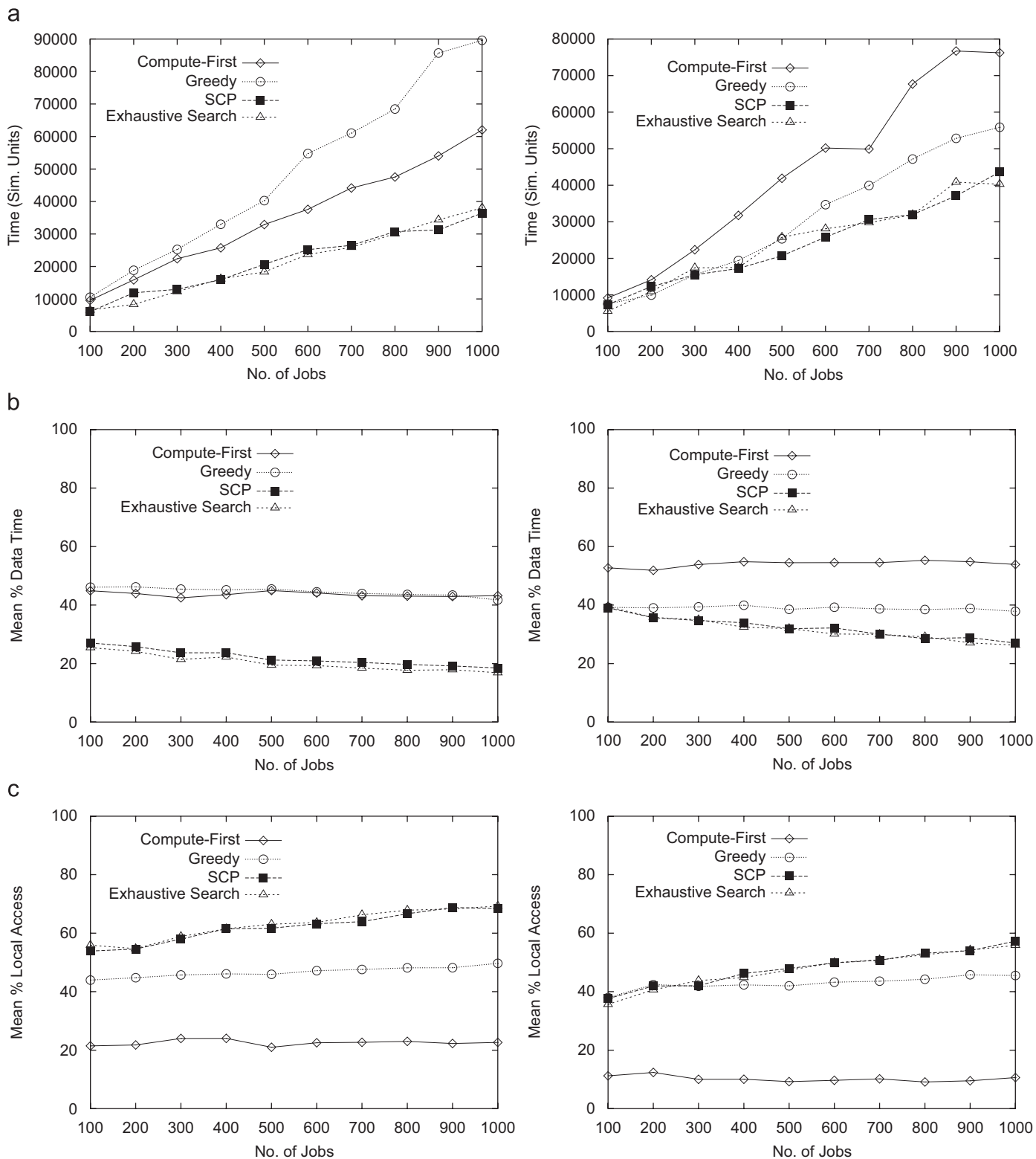


Fig. 8. Evaluation with increasing number of jobs ($Size = 300\,000\text{ MI}$, $K = 3$, left: $Dist = Uniform$, right: $Dist = Zipf$).

means that a job submitted to a compute resource is less likely to find its required data sets locally. This can be attributed to the rarer availability of data sets in Zipf distribution than in the Uniform distribution.

An interesting result here is that even with a high locality of access, the Greedy heuristic performs significantly worse than Compute-First for Uniform distribution (left column) while it performs better than the latter when the data sets are replicated

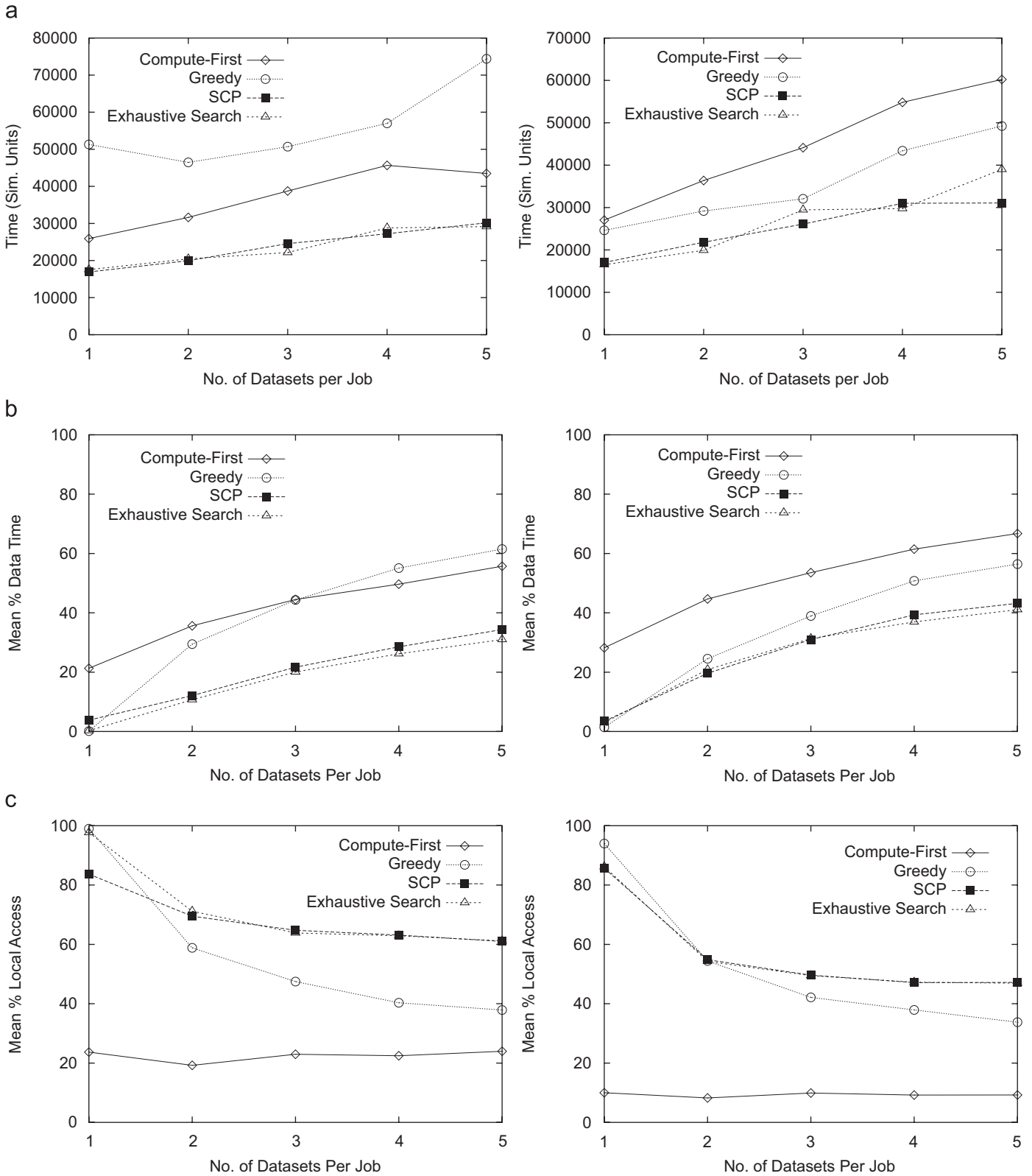


Fig. 9. Evaluation with increasing number of data sets per job ($N = 600$, $Size = 300000$ MI, left: $Dist = Uniform$, right: $Dist = Zipf$).

according to Zipf distribution. In the second case, there is a lower number of choices than in the first and thus, the Greedy strategy has a better probability of forming good resource sets.

In this case, it can be seen that the performance of Greedy comes close to or in some cases, becomes as competitive as SCP mirroring the results of Table 2. With a higher number of

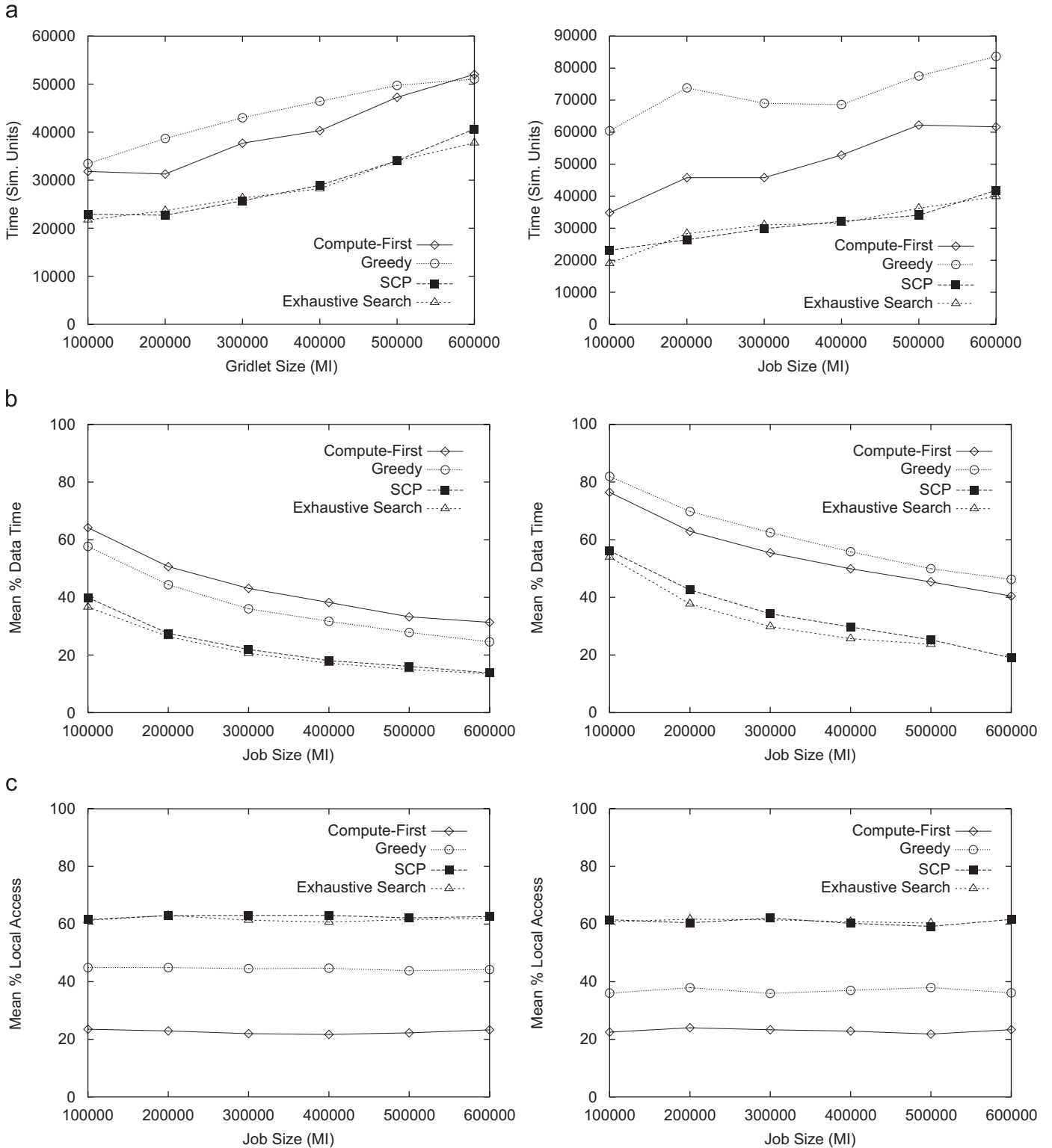


Fig. 10. Evaluation with increasing computational size ($N = 600$, $Dist = Uniform$, left: $K = 3$, right: $K = 5$).

choices, the Greedy strategy has a lower probability of arriving at the best compute resource for a job and its performance is degraded.

Fig. 9 shows the impact of changing only the number of data sets per job. Some of the trends in the previous graphs are also reflected here. With only one data set per job, all algorithms

except for Compute-First are able to produce schedules with zero data time and full locality of access. With the jobs per data set increasing, the impact of data transfer time increases at a faster rate for Greedy than for SCP and Exhaustive Search. Also, the locality reduces more steeply in the Zipf distribution than in the Uniform distribution, because there are fewer

Table 3
Summary of comparison between MinMin and Sufferage

Heuristic	Geometric Mean	Avg. deg	Avg. rank
<i>MinMin</i>			
Compute-First	19,604.73	18.7 (12.84)	4.93 (1.0)
Greedy	25,782.28	57.93 (28.51)	6.33 (1.45)
SCP	17,353.87	5.2 (13.58)	1.73 (1.44)
Exhaustive search	18,481.26	11.83 (11.39)	3.47 (1.41)
<i>Sufferage</i>			
Compute-First	60,631.56	269.31 (57.81)	8.0 (0)
Greedy	18,558.61	12.06 (8.45)	4.2 (1.72)
SCP	17,353.87	5.2 (13.58)	1.73 (1.44)
Exhaustive search	18,584.88	12.47 (11.53)	3.67 (1.53)

data hosts for each data set. Finally, Fig. 10 shows the impact of the computation time on the performance of data-oriented scheduling algorithms. The locality remains almost constant throughout the experiments. However, as expected, the impact of data transfer is steadily reduced with increasing size of computation.

Another interesting result here is that the performance of Exhaustive Search is worse than that of SCP in certain cases. This runs contrary to expectations that Exhaustive Search will produce the best results in every case. This is due to the fact that MinMin itself is not guaranteed to give the best schedules in every situation [29]. The assignment of resources to a job impacts the selection of resources for jobs that are yet to be assigned. This leads to variations in performance of all the algorithms.

7.2. Comparison between MinMin and Sufferage

Each of the matching heuristics were paired with both MinMin and Sufferage scheduling algorithms and evaluated to determine if the latter provided a better performance than the former. The results of the experiments carried out within this evaluation is summarised using the same metrics as in the previous section and are listed in Table 3. It can be seen that there is little difference in the performance of both SCP and Exhaustive Search heuristics when coupled with either MinMin or Sufferage scheduling algorithms. Also, there is only a slight improvement in the performance for Greedy when coupled with the Sufferage algorithm. However, the performance for Compute-First is significantly degraded by coupling it with the Sufferage algorithm. On average, it is about $2\frac{1}{2}$ times as worse as the best heuristic in any experiment. Also, the Compute-First-Sufferage pair is ranked 8th in terms of performance in all experiments (standard deviation is zero). In other words, it gives the worst performance in every case.

8. Related work

There has been a lot of work in scheduling interdependent tasks with communication dependencies and an overview of strategies static allocation of such tasks can be found in a survey published by Kwok and Ahmad [25]. One such strategy

proposed by Kafil and Ahmad [21] adapts the well-known A* search algorithm to search the entire space of possible task-processor mappings to identify one that provides optimal allocation with respect to processor load and task intercommunication. As mentioned previously, the work in this paper deals with a different model consisting of data-intensive independent (non-communicating) tasks where the tasks are not only mapped to processors but to storage resources as well. The matching algorithms also perform a bounded search within the task-resources (both compute and storage) mapping space but on a per task basis. Considering the entire space of all task to compute and data resource mappings will make the problem computationally intractable and hence, we have opted for a 2 stage mapping process.

Previous publications in scheduling distributed data-intensive applications on Grids [3,32,34] have tackled the problem of replicating the data for a single job depending on the site where the job is scheduled. However, the application model applied here is closer to that of Casanova et al. [11] who investigate scheduling algorithms for a set of independent tasks that share files. They extend the MinMin and Sufferage algorithms to consider data requirements of the tasks and introduce the XSufferage algorithm to take advantage of file locality. However, in their article the source of all the files for the tasks is the resource that dispatches the jobs. This work is extended by Giersch et al. [18] to consider the general problem of scheduling tasks that share multiple files, each available from multiple sources. They focus on developing routing algorithms for staging the input files through the network links on to data resources, close to the selected compute resources, such that the total execution time is minimised. Khanna et al. [23] propose a hypergraph-based approach for scheduling a set of independent tasks with a view to minimise the I/O overhead by considering the sharing of files between the tasks. However, they do not take into account the aspect of data replication as the files have only a single source.

The scheduling model considered in this paper is distinct from those mentioned previously because it considers: (a) the problem of selecting a resource set for a job requiring multiple data sets in an environment where the data is available from multiple sources due to prior replication and (b) the selection of computational and data resources in such a resource set to be interconnected. This paper also extends MinMin and Sufferage algorithms similar to that done by Casanova et al. [11] and Giersch et al. [18]. However, in the algorithms presented in this paper, the focus of the effort remains on matching or selection of resources which has not been given adequate weightage in related work. The matching algorithms aim to select a resource set such that both the computational and data transfer components of the execution time are reduced simultaneously. This is different from the approach, followed by most of the Data Grid scheduling algorithms of scheduling the jobs onto a compute resource based on minimum execution time and then replicating the data to minimise the access time. The latter approach was generalised and extended to support the multiple data sets model in the previous sections, and was evaluated as the Compute-First heuristic.

Simulation results show that Compute-First produces worse schedules when compared to a strategy giving weightage to both computational and data factors such as the SCP Tree Search algorithm.

Mohamed and Epema [30] present a Close-to-Files algorithm for a similar application model, though restricted to one data set per job, that searches the entire solution space for a combination of computational and storage resources to minimise execution time. This strategy, extended to support multiple data sets per job and evaluated as Exhaustive Search in the previous section, produces good schedules but becomes unmanageable for large solution spaces that occur when more than one data set is considered per job.

Jain et al. [20] proposed a set of heuristics for scheduling I/O operations so as to avoid transfer bottlenecks in parallel systems. However, these heuristics do not consider the problem of scheduling computational operations and also, the problem of selecting data sources in case of data replication. Other publications in parallel I/O optimisation [1,36,42] pay attention to improving performance through techniques such as interleaving and disk striping. However, such optimisation techniques are not the focus of this paper.

9. Conclusion and future work

This paper presents the problem of mapping an application with a collection of jobs that require multiple data sets that are each multiply replicated, to compute resources and data hosts in a Grid. It models the problem of matching the jobs as an instance of the SCP and proposes a tree-search heuristic based on a solution to the SCP. This is then combined with the MinMin and Sufferage algorithms for scheduling sets of independent jobs and evaluated through simulation against other matching heuristics such as Compute-First, Greedy Selection and Exhaustive Search. Experiments show that the SCP and the Exhaustive Search heuristics provide the best performance among all the four heuristics mainly because they exploit the locality of data sets, and thereby reduce the amount of data transferred during execution. However, the high computational complexity of Exhaustive Search means that it will search through large spaces that may become infeasible for jobs requiring large number of data sets. Also, the experimental results show that there is no gain in performance by applying the Sufferage heuristic in place of MinMin for scheduling the entire set of jobs.

As part of immediate future work, it is planned to evaluate the SCP mapping heuristic using other task scheduling algorithms such as Max-min and Genetic Algorithms. It would also be interesting to explore scheduling of distributed data-intensive tasks where they are interdependent. In this case, the overall mapping should not only take into account the location of distributed data but also the communication between the tasks. In present-day Grids, scientific applications are increasingly being composed as data-intensive workflows involving tasks that process, share and manage large, distributed data sets [15]. These workflows are generally modelled as Directed Acyclic Graphs (DAGs) and many scheduling strategies for

interdependent tasks have been applied for mapping workflows on to Grid resources [4,38]. A possible extension to the work presented in this paper would be to use the SCP search heuristic with a known DAG scheduling algorithm such as the Dynamic Critical Path (DCP) [24] to schedule workflows with distributed data-intensive tasks.

The scheduling strategies in this paper are considered to be conventional in the sense that the compute resources and data hosts serve all requests and accept all jobs regardless of their source, and the scheduling is driven by the need to improve traditional parameters of performance such as application throughput. However, the emerging economy-based model of Grids [7] considers resource providers to be independent agents that are incentivised by profit motives to contribute resources to a Grid. A consumer in this environment would have a limited budget and would therefore, aim to execute her application at resources that provide her the best service within her budget. In such an environment, both resource providers and consumers aim to improve their utility that may depend on non-system-centric metrics. Recent publications by Kwok et al. [26], and Khan and Ahmed [22] model interactions between the participants in an economy-based Grid as games and analyse the behaviour of different agents under different game-theoretic strategies. These have been performed from a computational Grid perspective. Extending this model to distributed data-intensive applications is also a possible future work.

References

- [1] A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J. Hollingsworth, J. Saltz, A. Sussman, Tuning the performance of I/O-intensive parallel applications, in: Proceedings of the Fourth Workshop on I/O in Parallel and Distributed Systems (IOPADS '96), ACM Press, Philadelphia, PA, USA, 1996.
- [2] E. Balas, M.W. Padberg, On the set-covering problem, *Oper. Res.* 20 (6) (1972) 1152–1161.
- [3] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockinger, F. Zini, Simulation of dynamic grid replication strategies in OptorSim, in: Proceedings of the Third International Workshop on Grid Computing (GRID 02), Springer-Verlag, Berlin, Germany, Baltimore, MD, USA, 2002, pp. 46–57.
- [4] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy, Task scheduling strategies for workflow-based applications in grids, in: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005), Cardiff, UK, IEEE CS Press, Los Alamitos, CA, USA, 2005.
- [5] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, R.F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *J. Parallel Distributed Comput.* 61 (6) (2001) 810–837.
- [6] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web catching and zipf-like distributions: evidence and implications, in: Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99), New York, NY, USA, 1999.
- [7] R. Buyya, Economic-based distributed resource management and scheduling for grid computing, Ph.D. Thesis, Monash University, Australia (2002).
- [8] R. Buyya, M. Murshed, GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, *Concurrency Comput. Practice and Experience (CCPE)* 14 (13–15) (2002) 1175–1220.

- [9] D.G. Cameron, R. Carvajal-Schiaffino, A.P. Millar, C. Nicholson, K. Stockinger, F. Zini, Evaluating scheduling and replica optimisation strategies in OporSim, in: Proceedings of the Fourth International Workshop on Grid Computing (Grid2003), IEEE CS Press, Los Alamitos, CA, USA, Phoenix, AZ, USA, 2003.
- [10] H. Casanova, Simgrid: a toolkit for the simulation of application scheduling, in: Proceedings of the First International Symposium on Cluster Computing and the Grid (CCGRID '01), IEEE CS Press, Los Alamitos, CA, USA, Brisbane, Australia, 2001.
- [11] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman, Heuristics for scheduling parameter sweep applications in grid environments, in: Proceedings of the Ninth Heterogeneous Computing Systems Workshop (HCW 2000), IEEE CS Press, Los Alamitos, CA, USA, Cancun, Mexico, 2000.
- [12] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, The data grid: towards an architecture for the distributed management and analysis of large scientific datasets, *J. Network Comput. Appl.* 23 (3) (2000) 187–200.
- [13] N. Christofides, *Graph Theory: An Algorithmic Approach*, Academic Publishers, London, UK, 1975, Ch. Independent and Dominating Sets—The Set Covering Problem, pp. 30–57, ISBN 012 1743350 0.
- [14] T.H. Cormen, C. Stein, R.L. Rivest, C.E. Leiserson, *Introduction to Algorithms*, McGraw-Hill Higher Education, 2001.
- [15] E. Deelman, et al., Mapping abstract complex workflows onto grid environments, *J. Grid Comput.* 1 (1) (2003) 25–39.
- [16] I. Foster, C. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, USA, 1999.
- [17] R. Gardner, et al., The Grid2003 production grid: principles and practice, in: Proceedings of the 13th Symposium on High Performance Distributed Computing (HPDC 13), IEEE CS Press, Los Alamitos, CA, USA, Honolulu, HI, USA, 2004.
- [18] A. Giersch, Y. Robert, F. Vivien, Scheduling tasks sharing files from distributed repositories, in: Proceedings of the 10th International EuroPar Conference (EuroPar '04), Springer-Verlag, Berlin, Germany, Pisa, Italy, 2004.
- [19] W. Hoschek, F.J. Jaen-Martinez, A. Samar, H. Stockinger, K. Stockinger, Data management in an international data grid project, in: Proceedings of the First IEEE/ACM International Workshop on Grid Computing (GRID '00), Springer-Verlag, Berlin, Germany, Bangalore, India, 2000.
- [20] R. Jain, K. Somalwar, J. Werth, J.C. Browne, Heuristics for scheduling I/O operations, *IEEE Trans. Parallel Distributed Systems* 8 (3) (1997) 310–320.
- [21] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, *IEEE Concurrency* 6 (3) (1998) 42–50.
- [22] S. Khan, I. Ahmad, Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation, in: Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Rhodes Island, Greece, IEEE CS Press, Los Alamitos, CA, USA, 2006.
- [23] G. Khanna, N. Vydyanathan, T. Kurc, U. Catalyurek, P. Wyckoff, J. Saltz, P. Sadayappan, A hypergraph partitioning-based approach for scheduling of tasks with batch-shared I/O, in: Proceedings of the 2005 IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005), IEEE CS Press, Cardiff, UK, 2005.
- [24] Y.-K. Kwok, I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors, *IEEE Trans. Parallel Distrib. System* 7 (5) (1996) 506–521.
- [25] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, *ACM Comput. Surveys* 31 (4) (1999) 406–471.
- [26] Y.-K. Kwok, S. Song, K. Hwang, Selfish grid computing: game-theoretic modeling and nas performance results, in: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), IEEE Computer Society, Washington, DC, USA, 2005, pp. 1143–1150.
- [27] I.C. Legrand, H.B. Newman, The MONARC toolset for simulating large network-distributed processing systems, in: Proceedings of the 32nd Winter Simulation Conference (WSC '00), Society for Computer Simulation International, San Diego, CA, Orlando, FL, 2000.
- [28] M. Maheshwaran, S. Ali, H.J. Siegel, D. Hengsen, R.F. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems, in: Eighth Heterogeneous Computing Systems Workshop (HCW '99), San Juan, Puerto Rico, 1999.
- [29] M. Maheshwaran, S. Ali, H.J. Siegel, D. Hengsen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, *J. Parallel Distributed Comput.* 59 (1999) 107–131.
- [30] H. Mohamed, D. Epema, An evaluation of the close-to-files processor and data co-allocation policy in multiclustes, in: Proceedings of the 2004 IEEE International Conference on Cluster Computing, IEEE CS Press, Los Alamitos, CA, USA, San Diego, CA, USA, 2004.
- [31] K. Park, G. Kim, M. Crovella, On the relationship between file sizes, in: Proceedings of the 1996 International Conference on Network Protocols (ICNP '96), IEEE CS Press, Atlanta, GA, USA, 1996.
- [32] S.-M. Park, J.-H. Kim, Chameleon: a resource scheduler in a data grid environment, in: Proceedings of the Third IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), IEEE CS Press, Los Alamitos, CA, USA, Tokyo, Japan, 2003.
- [33] A. Rajasekar, M. Wan, R. Moore, MySRB & SRB: components of a data grid, in: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), IEEE CS Press, Los Alamitos, CA, USA, Edinburgh, UK, 2002.
- [34] K. Ranganathan, I. Foster, Decoupling computation and data scheduling in distributed data-intensive applications, in: Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC), IEEE CS Press, Los Alamitos, CA, USA, Edinburgh, UK, 2002.
- [35] K. Ranganathan, I. Foster, Simulation studies of computation and data scheduling algorithms for data grids, *J. Grid Comput.* 1 (1) (2003) 53–62.
- [36] K. Salem, H. Garcia-Molina, Disk striping, in: Proceedings of the Second International Conference on Data Engineering (ICDE-86), IEEE CS Press, Los Alamitos, CA, USA, Los Angeles, USA, 1986.
- [37] E. Seidel, G. Allen, A. Merzky, J. Nabrzyski, GridLab: a grid application toolkit and testbed, *Future Generation Comput. Systems* 18 (8) (2002) 1143–1153.
- [38] Z. Shi, J.J. Dongarra, Scheduling workflow applications on processors with different capabilities, *Future Generation Comput. Systems* 22 (6) (2006) 665–675.
- [39] A. Sulistio, U. Cibej, B. Robic, R. Buyya, A tool for modelling and simulation of data grids with integration of data storage, Replication and Analysis, Technical Report GRIDS-TR-2005-13, University of Melbourne, Australia, November 2005.
- [40] A. Sulistio, G. Poduval, R. Buyya, C.-K. Tham, On incorporating differentiated network service into GridSim, Technical Report GRIDS-TR-2006-5, The University of Melbourne, Australia, March 2006.
- [41] A. Sulistio, C.S. Yeo, R. Buyya, A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools, *Software Practice and Experience (SPE)* 34 (7) (2004) 653–673.
- [42] R. Thakur, A. Choudhary, R. Bordawekar, S. More, S. Kuditipudi, Passion: optimized I/O for parallel applications, *Computer* 29 (6) (1996) 70–78.
- [43] S. Venugopal, R. Buyya, A deadline and budget constrained scheduling algorithm for e-science applications on data grids, in: Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005), Lecture Notes in Computer Science, vol. 3719, Springer-Verlag, Berlin, Germany, Melbourne, Australia, 2005.
- [44] S. Venugopal, R. Buyya, L. Winton, A grid service broker for scheduling distributed data-oriented applications on global grids, in: Proceedings of the Second Workshop on Middleware in Grid Computing (MGC 04), ACM Press, New York, USA, Toronto, Canada, 2004.
- [45] R. Wolski, N. Spring, J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, *J. Future Generation Comput. Systems* 15 (1999) 757–768.
- [46] N. Yamamoto, O. Tatebe, S. Sekiguchi, Parallel and distributed astronomical data analysis on grid datafarm, in: Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004), IEEE CS Press, Los Alamitos, CA, USA, Pittsburgh, USA, 2004.

Dr. Srikumar Venugopal is a Post-Doctoral Research Fellow in the Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, University of Melbourne, Australia. He received B.Tech. degree from Cochin University of Science and Technology, India in 2001 and Ph.D. from the University of Melbourne in 2006. His research interests are in resource allocation in large-scale distributed systems and data-intensive Grid computing.

Dr. Rajkumar Buyya is an Associate Professor and Reader of Computer Science and Software Engineering; and Director of the Grid Computing and

Distributed Systems (GRIDS) Laboratory at the University of Melbourne, Australia. He received B.E. and M.E. degrees from Mysore and Bangalore Universities in 1992 and 1995 respectively; and Doctor of Philosophy (Ph.D.) from Monash University, Melbourne, Australia in 2002. He has co-authored over 180 publications. He has co-founded and chaired four IEEE/ACM international conferences: CCGrid, Cluster, Grid, and E-Science. He has presented over 140 invited talks on his vision on IT Futures and advanced computing technologies in several international conferences and institutions in Asia, Australia, Europe, North America, and South America. For further information, please visit: <http://www.buyya.com>.