DECO: Data Replication and Execution CO-scheduling for Utility Grids

Vikas Agarwal, Gargi Dasgupta, Koustuv Dasgupta,
Amit Purohit, and Balaji Viswanathan

IBM, India Research Lab
{avikas, gdasgupt, kdasgupta, ampurohi, bviswana}@in.ibm.com

Abstract. Vendor strategies to standardize grid computing as the IT backbone for service-oriented architectures have created business opportunities to offer grid as a utility service for compute and data–intensive applications. With this shift in focus, there is an emerging need to incorporate agreements that represent the QoS expectations (e.g. response time) of customer applications and the prices they are willing to pay. We consider a utility model where each grid application (job) is associated with a function, that captures the revenue accrued by the provider on servicing it within a specified deadline. The function also specifies the penalty incurred on failing to meet the deadline. Scheduled execution of jobs on appropriate sites, along with timely transfer of data closer to compute sites, collectively work towards meeting these deadlines. To this end, we present DECO, a grid meta-scheduler that tightly integrates the compute and data transfer times of each job. A unique feature of DECO is that it enables differentiated QoS – by assigning profitable jobs to more powerful sites and transferring the datasets associated with them at a higher priority. Further, it employs replication of popular datasets to save on transfer times. Experimental studies demonstrate that DECO earns significantly better revenue for the grid provider, when compared to alternative scheduling methodologies.

1 Introduction

Grid computing is a form of distributed computing in which the use of heterogeneous resources (computation, storage, applications and data), spread across geographic locations and administrative domains, is optimized through virtualization and collective management. Initially conceived to support compute–intensive scientific applications and to share massive datasets, enterprises of all sizes and shapes are slowly beginning to recognize the technology as a foundation for management of IT resources, enabling them to better meet business objectives. Rapid advances in Web services technology have further provided an evolutionary path from the "stovepipe" architecture of traditional grids to a standardized, service–oriented, enterprise class grid of the future. The convergence of SOA and grid computing is embodied by the Global Grid Forum's Open Grid Services Architecture (OGSA) [1] that describes a service–oriented

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 52–65, 2006.

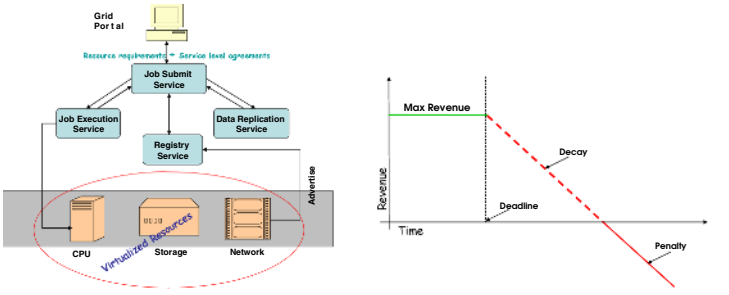
grid. The specifications provide various capabilities like job execution, data management, resource management, and security, some of which are illustrated in Fig.1(a).

It is of little surprise that the IT infrastructure of the future is being dubbed as the network grid, a universal computing paradigm that takes the ubiquitous connectivity of the Internet to its next logical step–ubiquitous *utility computing*. Grid providers will now store and manage customer data and run their applications in exchange of a usage fee. Today, a number of major vendors are advocating utility computing– for example Sun Grid, HP Utility Data Center, and IBM Deep Computing Capacity On Demand offerings, all promise its customers the choice and control on how to purchase and leverage IT power for competitive advantage. What is interesting about this computing model is its fiscal impact. With the utility grid, IT dollars and resources are not tied up in hardware and administrative costs. Instead, the focus shifts to the more strategic aspects of IT, such as Service Level Agreements (SLAs). These agreements specify QoS–based pricing policies for applications requiring access to computation referred to as compute in rest of the paper and data resources, and enable grid customers to delineate and prioritize business deliverables. From the provider's point of view, SLAs provide endless possibilities for business revenue maximization, based on differentiated quality–of–service.

1.1 Towards a Utility Framework for Grids

In a service-oriented grid, several heterogeneous cluster sites are inter-connected by WAN routers and links. Each cluster site is associated with some compute power and some storage space. The grid hosts customer data and provides compute capabilities. It charges each customer application (job) for usage of compute and storage resources. The conditions for service payments can be captured by a utility model that guarantees a certain QoS level for the price associated with it. Utility models have been proposed for scheduling and resource management in computational grids [2,3].

Since the response time of an application (job) serves as a common service–level objective for providers, we consider a *revenue function* illustrated in Fig.1(b), that captures the QoS requirements in terms of the response time expected by the job and the price it is willing to pay for it. If a job finishes within a deadline T, then the provider earns a revenue R. Otherwise, the revenue decays linearly at a constant rate δ. Eventually, the revenue may decay to a negative number, indicating a *penalty*. The penalty may or may not be unbounded. Each job submitted to the grid is associated with a revenue function. The revenue functions capture a rich space of policy choices by capturing the importance of a task (maximum revenue) as well as its urgency (decay) as separate measures. Each job has some compute requirements in terms of CPU time, and some data requirements, in terms of data files (objects) required for the computation. The total response time of a job is dependent on its execution time, as well as the time taken to transfer its data to the compute site. The execution time depends on the nature of the job and the compute power of the site assigned to it. The



(a) High–level architecture of service–oriented grids (b) Revenue function for utility grids

Fig. 1. Utility framework for grids

transfer time depends on the network connectivity and the location of the data objects. Because of the large number and size of data objects, it is unlikely that a site will have all the data required to execute any job. For data–intensive jobs, the time taken to complete the data transfers to the compute sites over WAN links, can thus be potentially significant. Hence, it is of critical importance that data objects are placed closer to the jobs accessing them. Further, a job needs to transfer all its data before it can execute. It is, however, possible to overlap the transfer time of a job with the execution of other jobs that have their data available locally. Finally, in a utility framework, the order in which the job executions and data transfers are scheduled have implications on the completion time of each job and, hence, the revenue earned by the provider.

Traditional grid solutions have inherently decoupled the execution of jobs from data transfer (and placement) decisions. The *Job Execution Service* handles the scheduling of a batch of jobs at different compute sites. The choice of a site for each job depends upon factors like load on the site, availability of datasets locally etc. Approaches for assigning job execution have been proposed in [4,5]. Multiple transfers of the same data object is avoided by creating replicas of the object at selected sites. The *Data Replication Service* of the grid provides this functionality. A number of algorithms have been proposed in the literature [6,7] for data replication in grids. In each case, changes in data placement are prescribed by a long–term replication process, that studies the history of accesses to data objects. Data objects are transferred (replicated) across sites using transfer protocols like GridFTP [8]. However, decoupling execution assignment from data transfer (and replication) often leads to poor and in-efficient response time for jobs. Since the finish time of jobs translates directly to dollars earned or lost, it is very critical to consider both the execution and transfer times of each job. To do so, job execution service needs to work in close co–ordination with the data replication service. Asynchronous replication of historically popular objects is no longer enough— placement decisions need to be based on the compute locations of jobs, and vice versa. Finally, it is imperative that job and data transfer scheduling policies incorporate service differentiation, i.e. jobs that have

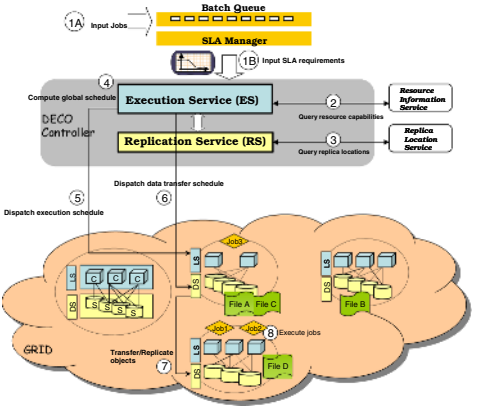


Fig. 2. DECO Architecture

higher revenues and/or harsher penalties need to be assigned higher priorities. This leads towards a *utility–based co–scheduling* framework for grids.

1.2 Contributions

We present **DECO**, a system for **D**ata replication and **E**xecution **CO**-scheduling in utility grids. DECO is designed for business revenue maximization of the grid service provider. DECO decides which job to assign to which site, which objects to replicate at which site, when to execute each job, and finally when to transfer (or replicate) data across the sites. A unique feature of DECO is that it enables differentiated QoS based on the revenue functions of the jobs and their compute/data requirements. Our main contributions include: (i) a co-scheduling framework that tightly couples the job execution and data replication services in a utility grid; (ii) a (meta)scheduling system that co-ordinates the placements of jobs and data objects, and (iii) a differentiated approach for scheduling job executions and data transfers (replications) aimed at maximizing the revenue earned by the provider.

2 DECO Architecture

We propose a co–scheduling framework for integrating the execution and data transfer times of compute and data–intensive applications in grids. Fig.2 gives a detailed view of the proposed framework. We consider as input a batch of grid jobs along with compute and data requirements, and SLA descriptions managed by an `SLA Manager`. Admission controller selects and places jobs in the `Batch Queue` based on business policies, current system state etc. The `DECO Controller` is a single point of submission for all jobs. It computes an offline schedule periodically (for e.g. every 24 hrs), for all unfinished jobs in the queue. The controller works on the following assumptions: every job needs to execute

at one cluster site; all the data objects needed by a job should be present at its execution site; jobs are independent and have no dependencies on other jobs.

The two primary components of the controller are the `Execution Service (ES)` and the `Replication Service (RS)`. There exists a tight integration between the functionalities of these components. The workflow of the controller is as follows: (1) ES gathers resource availability information from a Resource Information Service. RS gathers location information from a Replica Location Service. (2) Depending on the utility values of jobs and the cost benefits obtained from replication, ES in conjunction with RS, advises job execution sites and replica creation activities of popular objects. (3) Once the decision is made on which jobs will execute where and what data is to be placed where, the controller uses its global view of the grid topology to compute a master schedule containing an ordered sequence of replication, transfer and execution events across clusters. (4) From the master schedule, DECO extracts the corresponding cluster-specific schedule and dispatches it to each cluster site. Finally, at each cluster site, there is a local job scheduler responsible for intra-cluster job scheduling and management of resources and a data scheduler responsible for handling data transfers to and from the site.

3 Utility–Based Data Replication and Execution Co-scheduling

Assume that the time horizon can be divided into L discrete time intervals, all not necessarily of equal length. If a job finishes in an interval within its completion time deadline T_j, it earns a revenue of Rev_j, else incurs penalty at the rate of Pen_j per hour. Let Z denote the net business revenue earned by all jobs in the workload. The overall approach for maximizing Z is summarized in two steps:

- **Step 1:** Weigh the reward of scheduling each job with the risk of delaying it. Based on these weights decide on which jobs to execute in which time interval and meet their completion time goals by (a) assigning them to appropriate cluster sites and (b) replicating their data objects.
- **Step 2:** Given the entire topology of the provider's network, determine a time schedule of when the job executions and the data transfers (replications) should begin/end.

3.1 Step 1: Integrated Job Assignment and Data Placement

Consider a set of M cluster sites, N data objects and K jobs. Each site i has an associated compute capacity A_i and a storage capacity S_i. Cluster sites a and b are connected by a WAN link of available bandwidth bw_{ab}. Each object o is of size s_o and is associated with a replica set R_o that specifies the clusters at which the object is currently placed. Each job j submitted to the batch queue specifies a compute requirement e_j, and a set of data objects F_j that it will operate on.

WAN transfer times are assumed to be dominant over the LAN parameters. Let β_{io} denote 1 if the data object o is replicated at site i, and 0 otherwise. For job j executing at cluster site i, let te_{ij} denote the job execution time, tr_{ij} denote the total transfer time of all objects in F_j that are not locally present at i, tr_{io} denote the transfer time of the data object o to site i, and bestReplica(i,o) $\in R_o$, denote the cluster site that holds the replica of o and is connected by the highest bandwidth link to i. Then the total time taken to execute job j at site i, t_{ij}, is given by

$$t_{ij} = te_{ij} + tr_{ij} \tag{1}$$

where

$$tr_{ij} = \sum_{o \in F_j} tr_{io} \tag{2}$$

and

$$tr_{io} = (1 - \beta_{io}) s_o / bw(i, bestReplica(i, o)) \tag{3}$$

Let α_{ijl} be an indicator variable denoting 1 if job j is assigned to site i and finishes execution in time interval l, and 0 otherwise. Let U_{jl} denote the utility value of job j finishing at time l. If $\alpha_{ijl} = 1$ and $l \leq T_j$, then $U_{jl} = Rev_j$, else $U_{jl} = Rev_j - (l - T_j) * Pen_j$. The optimal assignment of jobs to sites and objects to sites such that Z is maximized, can be found by solving for the α and β assignments in the following program:
Maximize

$$T = \sum_{i=1}^{M} \sum_{j=1}^{K} \sum_{l=1}^{L} \alpha_{ijl} U_{jl} \tag{4}$$

subject to

Feasibility constraint:

$$\sum_{i=1}^{M} \sum_{l=1}^{L} \alpha_{ijl} = 1, \quad \forall j \tag{5}$$

Compute execution constraint:

$$\sum_{p=1}^{l} \sum_{j=1}^{K} \alpha_{ijp} (te_{ij} + \sum_{o \in F_j} tr_{io}) \leq A_i l, \quad \forall i, l \tag{6}$$

The feasibility constraint ensures that each job finishes in exactly one time interval at a site. The compute execution constraint makes sure that the number of jobs that can complete in time l at a site is atmost A_i times l. The above problem is Max-SNP hard[9] and hence is difficult even to approximate. To obtain a solution we present a simplification of the above problem. We begin with an initial placement $\hat{P}$ of data objects and consider the following two sub–problems: **AssignJobs**:- given the placement of objects, solve the problem for optimal assignment of jobs to cluster sites that maximizes the earned revenue. This reduces to solving the above problem in eqn. 4 for the α variables only (with β equal to 0 or 1). This problem is NP-hard and we design a linear-relaxation

based heuristic for solving it. We relax α to take real values and then round the α values to nearest integers. **AssignReplicas**:- given the job assignments, determine an optimal assignment of replicas to sites such that the replication benefit is maximized. The benefit for replication considered here is the increase in total business profit due to creation of a replica. Consider a replica of object o created at site i. From among the set of jobs assigned to site i and incurring penalties, some of them will be able to now meet their deadline. For each such job j that was paying a penalty of Pen_j, the increment in revenue obtained is $Rev_j - Pen_j$. Let c_{io} denote the total increment in revenue obtained by placing a replica of o at site i. The goal is determine additional replicas of objects such that the increase in business revenue is maximized:
Maximize

$$\sum_{i=1}^{M} \sum_{o=1}^{N} \beta_{io} c_{io} \tag{7}$$

subject to storage constraint:

$$\sum_{o=1}^{N} \beta_{io} s_o \leq S_i, \quad \forall i \tag{8}$$

We relax β to take real values and solve the linear program. The β values returned by the LP are always integral and hence the solution is optimal.

The combination of steps **AssignJobs** and **AssignReplicas** returns an approximate solution to the integrated job assignment and data placement problem. To bring the solution closer to the optimal, we suggest an iterative approach where the placement assignments from **AssignReplicas** serve as input to step **AssignJobs** and the procedure is repeated for k iterations. After the final iteration, an assignment of jobs and data objects to sites is obtained along with the start-time d_j for each job execution at cluster site i.

3.2 Step 2: Computing the Master Schedule

In this step, we derive the time to replicate objects across sites and determine a master schedule that specifies 1) selection of source replicas from which to initiate transfers/replications, and 2) computing a master schedule that specifies when (and where) all data transfers and job executions occur.

Selection of source replicas and computing master schedule. As before, we consider the time horizon can be divided into discrete time intervals, all not necessarily of equal length. Let t_{io} denote the time taken to transfer object o on link i. Let x_{iol} be 1 if transfer of object o on link i completes in the time interval $(l-1, l]$. The deadlines before which each data transfer should complete is given by d_o. These deadlines represent the maximum time before which the object o needs to be present at site i, and are obtained from the start time of jobs (d_j) in job execution schedule obtained from the final iteration of Step 1. When object o is required by multiple jobs with different start times, the earliest

start time (minimum d_j) represents the deadline d_o. The problem of selecting source replicas such that these deadlines are satisfied is then essentially finding a feasible assignment of x_{iol} subject to the constraints:

$$\sum_{i=1}^{TotalLinks} \sum_{l=1}^{L} x_{iol} = 1, \quad \forall o \tag{9}$$

$$\sum_{o=1}^{N} \sum_{k=1}^{l} x_{iok} t_{io} \leq l, \quad \forall i, \forall l \tag{10}$$

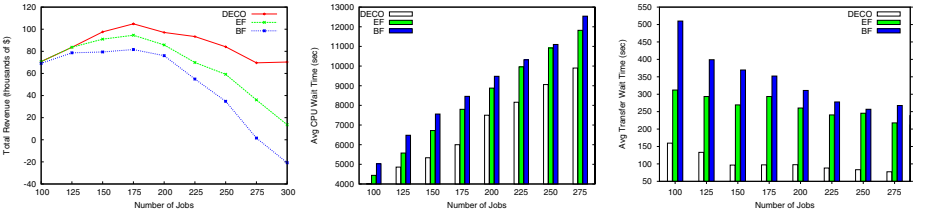
$$\sum_{i=1}^{TotalLinks} \sum_{l=1}^{L} x_{iol} l \leq d_o, \quad \forall o \tag{11}$$

Eqn. 9 makes sure that a transfer finishes in exactly one time interval on one link; eqn. 10 ascertains that there is no more than one simultaneous transfer on a link; eqn. 11 guarantees all transfers finish before their respective job start times. The resulting LP is essentially the one considered by Shmoys [9] and is known to be NP-hard. We use a rounding heuristic to solve the problem in polynomial time based on the one outlined in [9]. The heuristic returns a selection of links on which to schedule the transfer such that the deadlines are met. Finally, we compute a master schedule that specifies the assignment and ordering of all job executions and replication/data transfer activities.

4 Performance Evaluation

In this section, we evaluate the performance of the proposed two–step algorithm and compare it with alternative heuristics. We use the GridSim toolkit with its new Data Grid capabilities [10], to simulate a compute and data–intensive grid environment. For evaluation, we use a network topology based on the EU DataGrid TestBed topology with 11 sites and 23 links . Each site in the topology is modeled as a cluster site with finite compute and storage resources. The resource settings are obtained from a real testbed scenario outlined in [10]. To make the simulation feasible, we scaled down the compute and storage capacities of all sites while ignoring a few sites with very low compute and network resources. The network link bandwidths are used as specified.

Utility model: The batch of jobs is divided into three classes of Gold, Silver and Bronze jobs depending on their response time requirements (deadlines) and the corresponding revenue they are willing to pay. These jobs also differ in the penalty charged for missing the deadline. Each job has a revenue function associated with it, as described in Section 1, with unbounded penalty. The maximum revenue for a job and its penalty rate are picked from normal distributions with mean dependent on the class that the job belongs to. For the set of results presented here, the ratio



(a) (b) (c)

Fig. 3. Performance of all algorithms (a) Total revenue earned as number of jobs increase (b) Average CPU wait time of jobs before they are granted CPU (c) Average transfer wait time of jobs before all their files are available

of the means for Gold, Silver and Bronze is 5:3:2. The mix of Gold, Silver and Bronze jobs is 20%, 60%, and 20% respectively.

Job distribution: Each job specifies an execution time and a set of files required to execute the job. The number of files per job follows a normal distribution with a variance of 0.25. We vary the mean of the distribution between 3 and 6 to simulate various data-intensive scenarios. To model a realistic workload, the files for a job are chosen based on a Zipf distribution . The execution time for each job is approximately 20 minutes $\pm$ 30%. The job deadline time is a factor of its execution time and number of file dependencies. In our experimental setting, gold jobs have smaller (and hence) stricter deadlines than Silver and Bronze jobs. The ratio of the deadlines for Gold, Silver and Bronze jobs is 3:5:7.

Data distribution: Data objects considered in the experiments have an average file size of 1 GB, where file sizes follow a power-law(Pareto) distribution. We begin with a random placement of files at the cluster sites, with each file placed at exactly one site. Files are replicated as long as storage space is available at a site. The file replacement policy of the storage manager at a site is assumed to be LRU.

4.1 Alternative Heuristics

Previous approaches exist that use heuristics for value-based scheduling in computation grids, however they do not consider data requirements. Similarly, approaches in traditional data grids for compute and data assignments do not have an associated utility notion. To obtain competitive alternatives, we combine the best known utility–based scheduling policy [2,3], with the best known heuristics for compute and data assignment.

For each job j delayed by l hours beyond its deadline T_j, its utility value is given by $U_j = (l * Pen_j)/T_j$, where Pen_j is the penalty rate. The list of jobs is sorted based on their utility values. We consider two compute assignment techniques, earliest fit (EF) and best fit (BF). Starting with the first job in the sorted list, EF assigns it to the site that can satisfy it at the earliest time interval. This is a greedy approach, based on the generic Min-Min algorithm

[4,5], that aims at finishing jobs as early as possible. On the other hand, BF assigns the job to the site that can satisfy it at the latest time interval but still meets its deadline [3]. This approach keeps earlier time slots free for later jobs in the list. Both EF and BF, first transfer the required non-local files before it begins execution.

Previous work [11,6] outlines a Threshold based replication scheme, where each site records the number of local data transfers it needs to do for a globally popular file. When number of local transfers exceed a specified threshold, a replica is created locally. This Threshold based scheme is used in EF and BF to replicate files at cluster sites. Note that, both these heuristics have loose coupling of data replication and job execution. We compare DECO's tightly coupled replication and job execution algorithm with the above heuristics. For all the algorithms, we measure the following: (a) total revenue earned by a batch of jobs, including the revenues from satisfied jobs and penalties from those missing their deadlines (b) average time a job has to wait in the queue before it begins execution (c) business revenue earned by creating replicas.

4.2 Experimental Results

In the first set of experiments, we compare the revenue earned by DECO in comparison with EF and BF as shown in Fig.3(a). We increase the number of jobs in a batch, while keeping the system resources constant. We note that for all the approaches, the revenue steadily increases reaching a *peak*, and then dips beyond a *knee* point. The resource contention is low in the beginning with fewer jobs entering the system. With increasing number of jobs, system utilization improves leading to higher revenue. The peak of the curve represents an optimal system state when the revenue earned is the maximum. Increasing jobs beyond this point leads to greater resource contention and causes more jobs to miss their deadlines. Continual increase in the delay incurred by jobs is indicative of an overloaded system state and leads to reduction in revenue. This is captured by the zone beyond the knee of the curve. In an under-utilized system state all algorithms achieve comparable performance. But as the number of jobs is increased, DECO shows sustained significant improvement over EF and BF. Even in high resource contention conditions, revenue drop of DECO is less as compared to the alternative heuristics. This is observed by the sharp fall in EF and BF beyond the knee point as opposed to graceful degradation of DECO. On an average, DECO has (30-40)% improvement in revenue earnings over alternative approaches.

Fig.3(b) and Fig.3(c) report the average wait time of jobs, divided into CPU wait time and transfer wait time, for increasing number of jobs. The first component represents the cpu utilization and captures benefits of scheduling some jobs ahead of others. The second component, represents data availability and captures the benefits of replication and transfer scheduling. CPU wait time reduces when a job is assigned to a compute site with low queue lengths. Similarly, transfer wait time reduces with smart scheduling and parallelizing data transfers with job executions. With a batch size of 200 jobs, DECO shows about

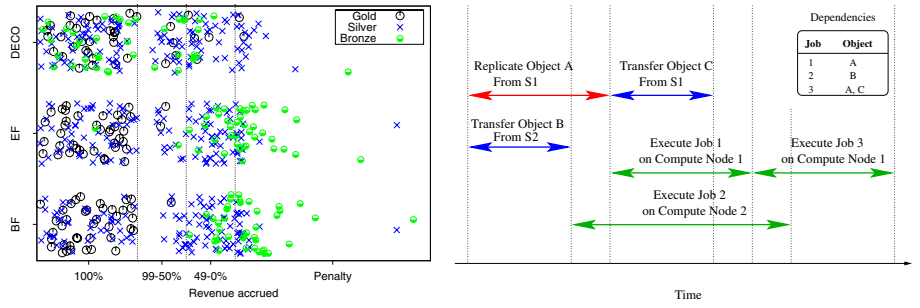


Fig. 4. (a) Scatter plot showing the revenue accrued by jobs (b) Illustration of master schedule

30% reduction in CPU wait time over EF and LF. With an average requirement of 5 files/job, DECO reduces the transfer wait time by as much as 40%.

Fig.4(a) sheds light on the superior performance of DECO over alternative heuristics. The experiment was conducted with a batch of 200 jobs. For each job, we determine the total response time and the corresponding revenue earned. (Note that, the revenue may be positive or negative depending on the scheduled time interval). The entire region is divided into four revenue zones. From left to right the vertical regions mean the following: the first vertical represents revenues earned by jobs that finish within the deadline. The second and third for jobs that miss their deadlines, but earn 50-99% and 0-49% of their maximum revenue respectively. The last contains jobs that incur penalties for finishing late.

Fig.4(a) reports the distribution of Gold, Silver and Bronze jobs in the four revenue zones. The alternative heuristics try to accommodate the heavy penalty incurring Gold and Silver jobs in the earlier time intervals, so as to avoid missing their deadlines. As a result most of the Bronze jobs are forced in the penalty zone and a significant number of Silver jobs earn less than a third of their maximum revenue. They fail to capture the possibility that the penalty from a large number of Bronze jobs can overshadow the revenue earned from Gold and Silver jobs. DECO intelligently chooses the right balance of placing the different types of jobs in suitable time intervals by weighing both their revenues and penalties, and thereby accrues significantly higher revenue than its competitors. Fig.4(b) show the actual data transfers, replications and executions scheduled by DECO along the timeline. The schedule is shown for three representative jobs (executing on the a compute site with two compute nodes). The jobs and their dependencies are as shown in Fig.4(b). In the schedule, replication of object A from site S1 is initiated along with transfer of object B from site S2. Jobs 1 and 2 begin execution in parallel with the transfer of object C from site S1. Job 3 however executes only after C's transfer completes and either of Job 1 or Job 2 finishes execution. This snapshot illustration demonstrates how DECO utilizes the available resources (network links, compute resources) to minimize idle time.

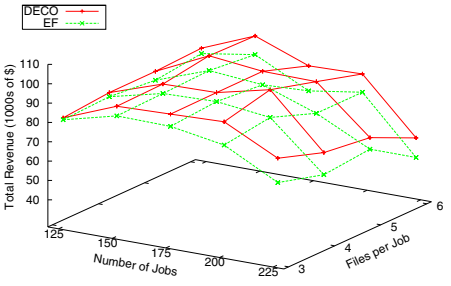


Fig. 5. Surface plot showing change in revenue as number of jobs and number of dependencies is varied

Table 1. Replica Utility showing Files per Job (FJ), Avg Replicas per File (AR) and Avg Replica Utility (ARU)

FJ	AR			ARU		
	DECO	EF	BF	DECO	EF	BF
3	1.3	1.1	1.1	99.7	43.1	35.3
4	1.4	1.2	1.3	88.0	68.6	53.5
5	1.5	1.3	1.3	116.3	62.8	60.0
6	1.7	1.4	1.4	130.2	65.7	61.3

In Fig.5, we compare the revenue earned as the number of jobs and their file dependencies are scaled for DECO and EF. When a job requires a larger set of files for its execution, job wait times are more pronounced as transfer times tend to be larger. The raised surface for DECO can be attributed to the fact that it creates more beneficial replicas when compared to EF, and achieves better parallelization of execution and data transfers. We additionally note that for 3–5 files per job, the revenue accrued by DECO steadily increases. Finally, for larger batch sizes and a larger number of file dependencies, the revenues begin to dip. This is because the system enters an overloaded state, as explained earlier.

Table-1 shows the replication statistics for each algorithm. We report the average number of replicas created per object, which captures the degree of replication. The table also reports the average utility of the replicated objects, which is defined as the revenue gain per new replica. Table–1 shows that the algorithms create 1.09–1.68 replicas per object, which is within acceptable limits for given storage constraints. Though more replicas are created by DECO, they contribute to significant revenue growth as reflected by the replica utility values. The replica utility of DECO is much better than the alternative heuristics(87.95–130.15 vs 43.14–68.55). This indicates that tight coupling of replication with compute allocation creates more meaningful replicas which are more effectively utilized while doing job assignment. Decoupled replication as adopted in the alternative heuristics fails to influence the job assignment directly, resulting in less optimal utilization of the replicas.

5 Related Work

Market–based economy models have recently received much attention in the grid utility computing domain [2,3]. [2] presents value-based scheduling heuristics that attempt to balance risk and reward of a job. [3] presents heuristics for admission control and resource allocation when jobs come with their own SLA requirements. However, both approaches do not consider the data transfer/replication as a part of the schedule. [12] presents a grid service broker for discovery of resources and

scheduling of jobs. Here, the scheduler minimizes the amount of data transfers by executing jobs at sites which have "nearer" access to data. The data placement however remains static. Resource allocation with failure provisioning for business profit maximization has been studied in [13]. However, co-ordinated placement of data with jobs has not been addressed. [14] describes a resource allocation approach for distributed computer systems based on competitive algorithms derived from Microeconomics. These algorithms however do not consider the problem of dynamic creation of replicas and its effect on auction pricing.

Among approaches that try to integrate job scheduling and data replication by incorporating network latencies and data transfer times are the Close-To-Files [4] and the Time-Budget constrained [5]. However, none of these approaches consider dynamic replication of data across sites. [6] looks at a number of techniques to dynamically replicate data across sites and assign jobs to sites. In this scheme, local monitors keep track of popular objects and preemptively replicates them at other sites. However, the work assumes homogeneous network conditions and single input files, both of which may not hold in case of globally distributed grids. [11,7] consider decoupled approach wherein the replication is controlled by an asynchronous process and looks at long-term history of access patterns. Stork [15] introduces a specialized scheduler for data placement activities and mentions the need for tighter integration between data placement and job execution. [16] presents an approach for co-scheduling by combining the Condor scheduler with a storage resource manager (SRM). Although, match-making is influenced by file placement, replication is performed without considering compute assignments. The co-scheduling problem addressed in [17] assumes single data object only. Finally, [18] solves the co-scheduling problem in a data grids using a genetic algorithm based heuristic. The work however, does not address the problem of optimal replica source selection and the time sequencing of job executions and replication/data transfers.

6 Conclusion

In this paper, we propose DECO, a co–scheduling framework for compute and data–intensive applications in utility grids. DECO employs a two–step algorithm for maximizing the business revenue of the grid provider. In Step 1, decisions are made on the assignment of jobs and replication of data objects. Step 2 schedules the data transfers (and replications) along with the job executions. The main highlight is that by integrating execution and data transfer times, DECO delivers significant improvements, both in terms of higher business revenues and lower job wait times, when compared to alternative approaches. As a part of future work, we will consider peer-to-peer approaches for resource-sharing among meta-schedulers. In this perspective, we plan to investigate distributed competitive algorithms [14]. Further, we are investigating application performance analysis techniques to help DECO make more up-to-date co-scheduling decisions. Finally, we plan to enhance the system with failure handling mechanisms.

References

1. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002)
2. Irwin, D.E., Grit, L.E., Chase, J.S.: Balancing risk and reward in a market-based task service. In: Proc. of HPDC'04. (2004)
3. Yeo, C.S., Buyya, R.: Service level agreement based allocation of cluster resources: Handling penalty to enhance utility. In: Proc. of Cluster 2005. (2005)
4. Mohamed, H., Epema, D.: An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In: Proc. of IEEE International Conference on Cluster Computing. (2004)
5. Venugopal, S., Buyya, R.: A deadline and budget constrained scheduling algorithm for e-science applications on data grids. In: Proc. of 6th International Conference on Algorithms and Architectures for Parallel Processing. (2005)
6. Ranganathan, K., Foster, I.: Computation scheduling and data replication algorithms for data grids. Grid resource management: state of the art and future trends (2004) 359–373
7. A. Chakrabarti, R.A.D., Sengupta, S.: Integration of scheduling and replication in data grids. In: Proc. of HiPC. (2004)
8. Allcock, W.: Gridftp protocol specification (global grid forum recommendation gfd.20). In: Globus Project: http://www.globus.org/alliance/publications/papers/GFD-R.0201.pdf. (2003)
9. Hall, L., Schulz, A., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: off-line and on-line algorithms. In: Proc. of ACM-SIAM Symposium on Discrete Algorithms. (1996)
10. Sulistio, A., Cibej, U., Buyya, R., Robic, B.: A toolkit for modeling and simulation of data grids with integration of data storage, replication and analysis. In: Technical Report, GRIDS-TR-2005-13, GRIDS Lab, University of Melbourne, Australia. (2005)
11. Ranganathan, K., Foster, I.: Decoupling computation and data scheduling in distributed data-intensive applications. In: Proc. of the 11 th IEEE International Symposium on High Performance Distributed Computing. (2002)
12. Venugopal, S., Buyya, R., Winton, L.: A grid service broker for scheduling distributed data-oriented applications on global grids. In: Proc. of the 2nd workshop on Middleware for grid computing. (2004)
13. Dasgupta, G., Dasgupta, K., Purohit, A., Viswanathan, B.: Qos-graf: A framework for qos based grid resource allocation with failure provisioning. In: Proc. of 14th IEEE IWQoS. (2006)
14. Ferguson, D.F., Yemini, Y., Nikolaou, C.: Microeconomic algorithms for load balancing in distributed computer systems. In: Proc. of ICDCS. (1988)
15. Kosar, T., Livny, M.: Stork: Making data placement a first class citizen in the grid. In: Proc. of the 24th Int. Conference on Distributed Computing Systems. (2004)
16. Romosan, A., Rotem, D., Shoshani, A., Wright, D.: Co-scheduling of computation and data on computer clusters. In: SSDBM'2005: Proceedings of the 17th SSDBM'2005. (2005)
17. H. Liu, M.B., Huang, J.: Dynamic co-scheduling of distributed computation and replication. In: Proc. of 6th IEEE Int. Symposium on Cluster Computing and the Grid (to appear). (2006)
18. Phan, T., Ranganathan, K., Sion, R.: Evolving toward the perfect schedule: Co-scheduling job assignments and data replication in wide-area systems using a genetic algorithm. In: Proc. of Job Scheduling Strategies for Parallel Processing. (2005)