Resource Management for Data Intensive Tasks on Grids

Imran Ahmad and Shikharesh Majumdar
Carleton University, Ottawa, Canada

1. Introduction

The ubiquitous Internet as well as the availability of powerful computers and high-speed network technologies as low-cost commodity components are changing the way computing is carried out. It becomes more feasible to use widely distributed computers for solving large-scale problems, which cannot often be effectively dealt without using a single existing powerful supercomputer. In terms of computations and data requirements, these problems are often resource intensive due to their size and complexity. They may also involve the use of a variety of heterogeneous resources that are not usually available in a single location. This led to the emergence of what is known as Grid computing. Grid computing enables sharing of heterogeneous distributed resources across different administrative and geographical boundaries [3]. By sharing these distributed resources, many complex distributed tasks can be performed in a cost effective way. The way the resources are allocated to tasks holds a pivotal importance for achieving satisfactory system performance [4]. To perform efficiently, the resource allocation algorithm has to take into account many factors, such as, the system and workload conditions, type of the task to be performed and the requirements of the end user.

To devise more efficient allocation algorithms, it may be useful to classify the given tasks into predefined types based on similarities in their predicted resource needs or workflows. This classification of tasks into various types provides the possibility to customize the allocation algorithm according to a particular group of similar tasks. This chapter presents an effective resource management middleware developed for a type of resource-intensive tasks classified as Processable Bulk Data Transfer (PBDT) tasks. The common trait among PBDT tasks is the transfer of a very large amount of data which has to be processed in some way before it can be delivered from a source node to a set of designated sink nodes (Ahmad, I & Majumdar, S. , 2008). Typically, these tasks can be broken down into parallel sub-tasks, called jobs. Various multimedia and High Energy Physics (HEP) applications can be classified as PBDT tasks. The processing operation involved in these tasks may be as simple as applying a compression algorithm to a raw video file in a multimedia application; or, as complex as isolating information about particles pertaining to certain wavelengths in High Energy Physics (HEP) experimentations [22][25]. Performing PBDT tasks requires both computing power and large bandwidths for data transmission. To perform such resource-intensive tasks, in recent years, research has been conducted in devising effective resource management middleware which has led to the creation of various efficient technologies. In order to provide a satisfactory performance, these systems must optimize the overall execution time (or makespan) of the resource-intensive tasks. This requires efficient allocation of the resources for the sub-tasks (called jobs in this paper) of the PBDT tasks at the individual machines of the network of nodes.

The problem of optimally scheduling these sub-tasks is a well-known NP complete problem [12]. To tackle it, various heuristics-based algorithms that can generate near-optimal solutions to optimization problems in polynomial times are devised. In this chapter a Bi-level Grid Resource Management System abbreviated as BiLeG is presented, in which the decision-making module is divided into two separate sub-modules. The upper level decision-making module is called the Task & Resource Pool Selector (TRPS). It selects a task from the given bag-of-tasks for which resources are to be assigned and chooses a partition of resources available for this chosen task (called the resource-pool of this task) which is typically a subset of all the resources available. The lower level decision-making module is called the Resource Allocator (RA), which uses an assignment algorithm to decide how the resources(from the chosen resource-pool) are allocated to the jobs, in a given task. Various algorithms can be used at RA whereas various policies can be deployed at TRPS. A particular combination of a TRPS policy and a RA scheduling algorithm deployed at a time is called an allocation-plan which determines the resource allocation for each task in the given bag-of-tasks. The following notation is used in this paper to write an allocation-plan: TRPS Policy, RA-Algorithm>. Investigating the choice of the most appropriate allocation-plan under a specific set of workload and system conditions is the focus of this chapter.

The main contributions of this paper are summarized.

1. It proposes the ATSRA algorithm and two extensions based on constraints relaxation technique.
   Based on simulation, it analyses the performance of the proposed algorithms for different number of available Grid nodes.
2. The experimental results capture the trade-off between accuracy in resource allocation and scheduling overhead both of which affect the overall system performance. The chapter discusses under which circumstances the proposed original algorithm or its extensions should be used.

The rest of the paper is organized as follows. In Section 2, different approaches to resource allocation of tasks on Grids are presented. In Section 3, PBDT tasks are described. In Section 4, the problem being solved is defined and an overview of the proposed system is presented. In Section 5 policies are described. In Section 6, the concept of Architectural Templates is described. In Section 7, a Linear Programming (LP) based algorithm and its extensions are described that can be used to perform PBDT tasks. In Section 8, experimental results are presented. Finally, in Section 9, the chapter is concluded.

2. Approaches to resource allocation of tasks on grids

Different researchers have taken various approaches to resource allocation of Tasks on Grids. The approaches to allocate resources in Grids can be divided into three broad categories.

1. Traditional Schedulers and Resource Brokers
2. Policy based Resource Allocation
3. Workflow based Resource Allocation

Each of these approaches is discussed in a following subsection.

2.1 Traditional schedulers and resource brokers

One of the traditional approaches is to use a Grid resource broker which selects suitable resources by interacting with various middleware services. Venugopal describes such a Grid resource broker that discovers computational and data resources running diverse middleware through distributed discovery services [12]. However, any mechanism for breaking a given task into parallel jobs for processing, is not present.

YarKhan and Dongarra [22] have also performed scheduling experiments in a Grid environment using simulated annealing. To evaluate the schedules generated by the simulated annealing algorithm they use a Performance Model, a function specifically created to predict the execution time of the program. Generating such a Performance Model requires detailed analysis of the program to be scheduled.

Another effort worth mentioning is Grid Application Development Software (GrADS) Project [2]. At the heart of the GrADS architecture is an enhanced execution environment which continually adapts the application to changes in the Grid resources, with the goal of maintaining overall performance at the highest possible level. A number of resource allocation algorithms can be used at GrADS to schedule a given bag-of-tasks in Grid environments. Due to the NP-complete nature of the resource allocation problem the majority of proposed solutions are heuristic algorithms [14] [18] [20].

2.2 Policy based resource allocation

For resource allocation in Grids, some researchers have also proposed policy based resource allocation techniques. Sander et al. [12] propose a policy based architecture for QoS configuration for systems that comprise different administrative domains in a Grid. They focus on making decisions when users attempt to make reservations for network bandwidth across several administrative network domains that are controlled by a bandwidth broker. The bandwidth broker acts as an allocator and establishes an end-to-end signalling process that chooses the most efficient path based on the available bandwidth. The work presented in [13] is concerned with data transmission costs only; whereas the research presented in this research needs to consider both computation and communication costs associated with the PBDT tasks. Verma. et al. [19] has also proposed a technique in which resource allocation is performed based on a predefined policy. But in this research, the resource allocation is not based on any performance measure.

2.3 Workflow based resource allocation

Many recent efforts have focused on scheduling of workflows in Grids. [16] presents a QoS-based workflow management system and a scheduling algorithm that match workflow applications with resources using event condition action rules. Pandey and Buyya have worked on scheduling scientific workflows using various approaches in the context of their

GridBus workflow management effort [11] [23]. [23] has developed an architecture to specify and to schedule workflows under resource allocation constraints. Also, many of the data Grid projects that support distributed processing of remote data have proposed workflow scheduling [11] [21].

3. Processable Bulk Data Transfer (PBDT) tasks

PBDT tasks require bulk transfer of processed data. Such data transfers are typical in multimedia systems and HEP experiments. For example in [1], 650MB of data was transferred on an average from a source to a set of sink nodes. High communication and computing times in PBDT tasks effectively amortizes the overhead of the LP-based algorithm used for optimization of the system performance. A PBDT task is characterized by the following three characteristics.

1. The task involves large data transfer that has to be processed in some way before it can be used at the sink nodes. The large amount of data involved in the PBDT differentiates it from the compute intensive tasks where data usually consists of is only the parameters of the remote functions invoked. This implies that the data communication costs cannot be ignored while scheduling a PBDT task.
2. Cost of data processing is proportional to the length of the raw data file.
3. The unprocessed raw file is such that it can be either processed as a whole or be divided into multiple partitions. If divided into partitions, each partition can be processed independently. The resultant processed partitions can later be combined to generate the required processed file. Consider a source file F, of size L. F can be partitioned into k disjoint partitions, with data sizes of $\{L_1, L_2 \ldots L_k\}$, such that

$$L = \sum_{i=1}^{k} L_i \tag{1}$$

Then for a PBDT task, the length of the required processed file is given by

$$L^{\sim} = \sum_{i=1}^{k} \varepsilon L_i \tag{2}$$

where ε_i is a processing factor which is the ratio of the size of the processed partition and that of the original partition.

PBDT tasks are increasingly becoming important. They are used in various multimedia, high-energy physics and medical applications. The following section explains two of the practical examples of PBDT tasks.

3.1 Particle physics data grids

Particle Physics Data Grids (PPDG) is a colloboratory project concerned with providing next-generation infrastructure for high-energy and nuclear physics experiments. One of the important requirements of PPDG is to deal with the enormous amount of data that is created during high-energy physics experiments that must be analyzed by large groups of specialists. Data storage, replication, job scheduling, resource management and security components of the Grid must be integrated for use by the physics collaborators. Processing these tasks require huge computing capabilities and fast communication capabilities. Grid computing is used for processing PPDG tasks that can be classified as a PBDT task.

3.2 Multimedia encoding

Multimedia encoding is required for applying a specific codec to a video [27]. Conventional methods use a single system for the conversion. The compression of the raw captured video data into an MPEG-1 or MPEG-2 data stream can take an enormous amount of time, which increases with higher quality conversions. Depending on the quality level of the video capture, the data required for a typical one hour tape can create over 10 GB of video data, which needs to be compressed to approximately 650 MB to fit on a VideoCD. The compression stage is CPU intensive, since it matches all parts of adjacent video frames looking for similar sub-pictures, and then creates an MPEG data stream encoding the frames. At higher quality levels, more data is initially captured and enhanced algorithms, which consume more time, are used. The compression process can take a day or more, depending on the quality level and the speed of the system being used. For commercial DVD quality, conversions are typically done by a service company that has developed higher quality conversion algorithms which may take considerable amount of time to execute. Grid technology is ideal for improving the process of video conversion.

4. Overall system architecture

In this research we have focused on the problem of allocating resources for a given bag of PBDT tasks. The bag-of-tasks consists of a set of independent PBDT tasks all of which must be executed successfully. The Grid system consists of n nodes. Collectively, these n nodes are represented by a set Δ. Each individual PBDT task in the given bag-of-tasks may be divided into a number of sub-tasks called *jobs* which can be executed in parallel, independent of each other. As discussed, PBDT tasks are resource-intensive tasks that use a large amount of computing resources and communication bandwidth. Usually, if a node starts processing a PBDT task, pre-emption of this task is counter-productive as it wastes the effort of transferring the raw-data file to the concerned node. Also, due to excessive demand of computing power, a node is assumed to handle the processing of only one PBDT task at a time. In this research we have made the following two assumptions regarding the running of constituent jobs of a task on Grid nodes.

1. Once a job starts executing on a Grid node, it cannot be pre-empted.
2. Only one job can be executed on a Grid node at a time.

For the cost analysis of the purposed architecture, we have measured *cost* by the time (in seconds) spent in performing a particular communication or computation job. We have chosen one megabyte as a unit of data. When a particular node i accesses data in node j, the communication cost of transporting a data unit from node i to node j is designated by d(i,j). It is assumed that the communication costs are metrics, meaning that they are non-negative, represented by Cp_m which is the cost of processing a unit of data. Set of all the nodes in the system is represented by **Δ**. To represent the computing costs, a vector of $|\mathbf{\Delta}|$ dimensions denoted by $[C_p]$ is used which holds the values of the computing costs of all the nodes in the system. A matrix $[C_c]$ of dimensions $|\mathbf{\Delta}|$ x $|\mathbf{\Delta}|$ denotes the values of the communication costs between all the nodes in the system. The objective of this research is to assign resources to tasks in such a manner that the total cost in executing the given bag-of-tasks is minimized; where the total cost is defined as the total time spent by a task at all the resources it has used during its execution. Total cost indicates the total resource usage for executing a task and hence the minimization of the total cost is a system objective.

The BiLeG resource management system consists of two decision-making modules; a lower level decision-making module called Resource Allocator (RA) and a higher level decision-making module called Task Resource Pool Selector (TRPS). TRPS selects a task T_i from the given bag of PBDT tasks and allocates it a resource-pool which is a subset of all resources available. A resource-pool of a particular task T_i is represented by Γ_i, where $\Gamma_i \subseteq \Delta$. RA allocates resources for a particular task T_i chosen from its associated resource-pool Γ_i.

Each PBDT task consists of an unprocessed raw-data file, information about the processing operation that is required to be performed on the raw-data file and a set of sink nodes where the processed file is to be delivered. The source node groups the submitted tasks into a bag-of-tsks (Fig. 1, Step-1) and initiates the processing by sending "initiate" signal to the TRPS Fig. 1 , Step-2). TRPS determines how many Grid resources are reserved (Fig. 1 , Step-3) by interacting with the Grid Computing Environment. This set of reserved nodes is represented by Δ. TRPS determines a resource-pool Γ_i for each of the tasks T_i. Not all the Grid nodes reserved are available or visible to an individual task T_i in the bag-of-tasks, T. Typically, each task has a different resource-pool selected by TRPS according to the TRPS policy used. For an individual task, using all the resources of the resource-pool may not be the best option for its most efficient execution. A TRPS resource selection policy is deployed at TRPS and determines the way in which TRPS chooses a resource-pool for each individual task. The policy uses the existing system state and resource availability in making its decision.

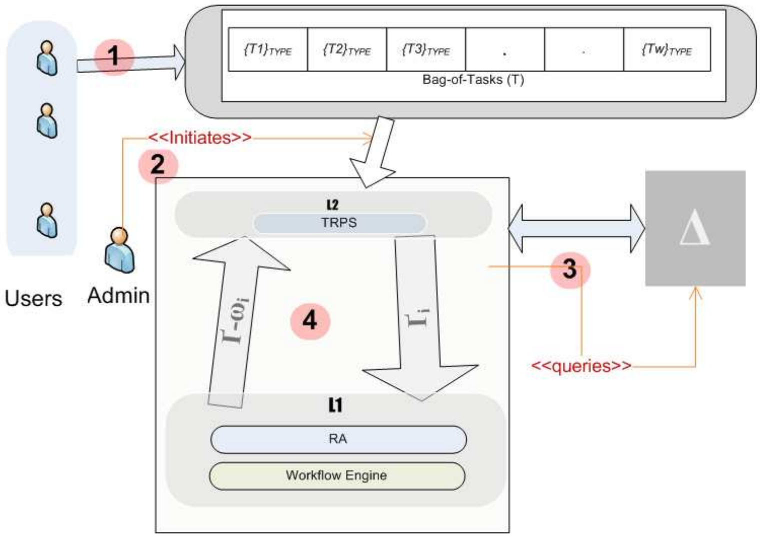


Fig. 1. BiLeG Architecture

From the resource-pool Γ_i allocated by TRPS to T_i, the lower level decision-making module (RA) chooses a set of resources that are used to perform T_i. This set of resources is denoted by ω_i. For different systems, different resource allocation algorithm may be best suited at RA. The remaining set of resources (Γ_i - ω_i) are returned to TRPS. Based on the resources chosen by the algorithm, RA divides a particular task into various jobs. RA specifies the details of the jobs in a file which is called the *workflow* of a Task, T_i. BiLeG architecture includes a software component called *workflow engine* which is designed to execute all the constituent jobs of a Task. The workflow engine is implemented as service and is responsible for running all the constituent jobs associated with a particular Task.

A combination of a TRPS policy and an RA algorithm is called an *Allocation Plan(AP)* and is represented by AP{<Policy>,<Algorithm>}. This paper explores the factors that determine the choice of the most efficient allocation plan for a given bag-of-tasks.

Note that the visibility of RA for a particular task is limited to its resource-pool. RA is myopic in nature and is not concerned with the overall system performance optimization. The objective of RA is to optimize the performance for a particular task only. TRPS is concerned with global system performance and has the responsibility to choose an appropriate resource-pool for each of the tasks and pass it on to RA. RA assigns a set of resources from the resource-pool passed to it by TRPS.

It can be observed that In the BiLeG architecture, by dividing the overall system into two independent decision-making modules and by assigning both decision-making modules separate responsibilities; we divide the problem of scheduling the tasks in the given bag-of-tasks into three different sub-problems:

1. Determination of the task execution order at TRPS
2. Selection of resource-pool
3. Resource allocation for each constituent job in the given bag-of-tasks at RA.

These three sub-problems may be solved by three independent algorithms. The division into three independent sub-problems makes the architecture customizable. It also provides finer-grade control over the resource allocation for the given bag-of-tasks and helps improving the stated optimization objective.

5. TRPS resoruce selection policy

A TRPS resource selection policy is used at the upper decision making module to select the resource-pool for each task. It can be either *static* or *dynamic* in nature. A TRPS policy is said to be static if mapping between tasks and their corresponding resource-pools is established before the system starts executing tasks and it is dynamic if these mappings are established during runtime according to the current availability of the resources. Two static TRPS policies considered in this paper are presented in this section. Dynamic TRPS polices are discussed available in [6].

5.1 Static Resource-Pool--Single Partition (SRP_{SP}) policy

In SRP_{SP}, the TRPS algorithm has two phases, a *mapping phase* and *an execution phase.*

The mapping phase (Fig. 2) is performed before the execution of the first task. Each task in T has a given resource-pool Δ. Thus, for each task $T_i \epsilon T$ and $\Gamma_i = \Delta$. In *Mapping* phase, a mapping between each task and the most appropriate set of resources it needs, is determined. To create this mapping, TRPS iteratively calls the algorithm at RA for each task in T.

In the *Execution Phase*, the first task in set T is executed first. TRPS iterates through all the tasks in T and chooses the next task for which the complete set of resources needed is available. All the tasks, for which each resource allocated by RA is available start executing. All the tasks, for which all the resources allocated by Resource Allocator are not yet available, wait in a queue. Once a task is complete, it is removed from T. The resources released by task are now added to the free resource set and the queue of waiting tasks is checked again to see whether the resource demand of any of these tasks can be satisfied. If all the resources of a particular are now available it starts execution and the next task in the waiting queues is checked and so on. The resources released by the task are now added to the resource set. The queue of waiting tasked is checked again in a First-In-First-Out (FIFO) order to see whether the resource demand of any of the tasks can be satisfied. When T={}, it means that all tasks have been assigned resources.

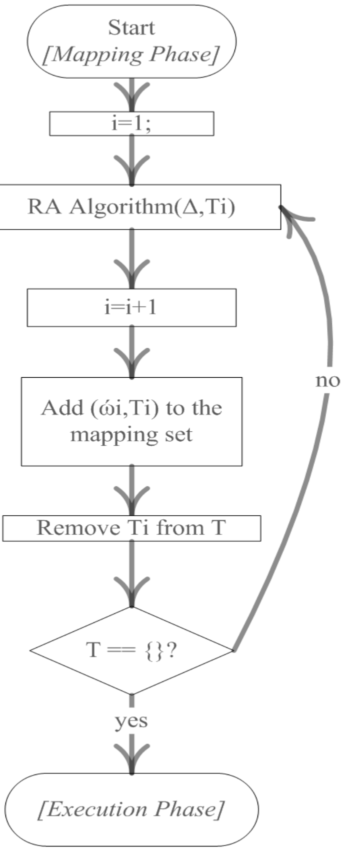


Fig. 2. Mapping Phase of Static Resource Pool Policies

5.2 Static Resource-Pool-Single Partition with Backfilling (SRP_{SP}+BF) policy

SRP_{SP}+BF is an improvement of SRP_{SP}. A drawback of SRP_{SP} is that the performance of the system may deteriorate due to two factors. First, there is the contention for resources, as each task has to wait until the complete set of resources it has been assigned to during the mapping phase becomes available. Second, there is the presence of unused resources that are not utilized at all; as it is possible that some resources may not become a part of mapping of any task. Thus, at a particular instance there may be resources that are available but are not being utilized while tasks are waiting in a queue (as the complete resource-pools associated with the tasks waiting in the queue are not available.)

The *mapping phase* of SRP_{SP}+BF is similar to SRP_{SP}. In the *execution phase*, SRP_{SP}+BF starts just like SRP_{SP}. Once all the tasks for which the resources are available have started to execute, the SRP_{SP}+BF tries to take advantage of the unused resource set by combining them into a single resource-pool. This resource-pool is given to the first task that is waiting in the queue and is it is passed to the Resource Allocator for the resource assignment. This process is called called *backfilling*. Backfilling is repeated till there is no unused resource in the system.

6. Architectural templates for RA

This section presents the concept of Architectural Templates that are used by the resource allocation algorithm deployed at the RA. In addition to deciding on how to decompose a task into its constituent jobs, an Architectural Template divides the available resources into different entities and assigns each of them a specialized role. We have identified the following five roles that can be assigned to the entities:

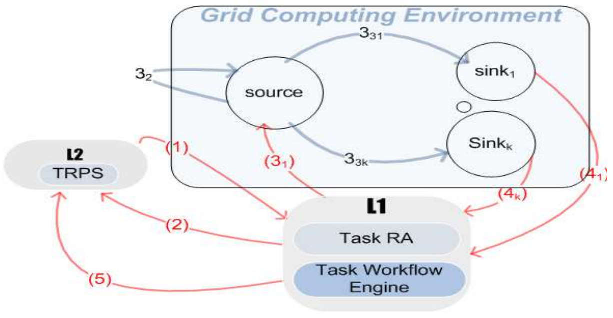
1. **Source:** A single Grid node where the raw data file of T_i is located.
2. **Sink:** A set of Grid nodes where the processed file is to be delivered.
3. **Compute-Farm:** A set of Grid nodes dedicated to process the raw data file in parallel.
4. **Data-Farm:** A set of Grid nodes used for replicating the data.
5. **Egress Node:** A node where files are combined after being processed at the compute-farm.

Note that a particular node may be part of a different entity at different times. For example a resource may be best utilized in a compute-farm for processing a particular job at one time, thus being a part of the compute-farm entity. But the same node may be used more effectively in a data-farm for processing a job in another task at another time; thus being a part of a data-farm entity. For each type of a task a set of appropriate templates is given as an input to the Resource Allocator. In this paper we have assumed the same set of templates described later can be used for every task in the bag-of-tasks. Thus, an Architectural Template specifies the structure of the suggested functional domains in which available resources are to be divided. This section briefly discusses a set of templates suitable for PBDT tasks.

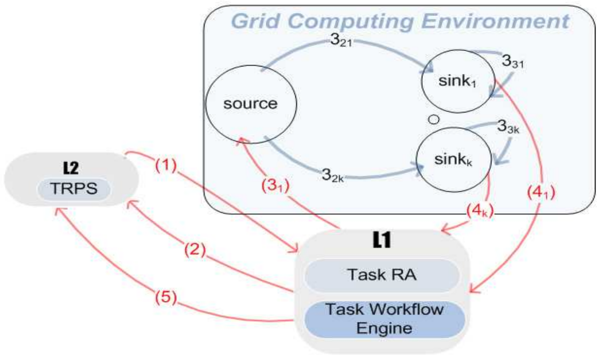
6.1 2-Tier Architectural Templates

In 2-Tier Templates only the source and the sink nodes are used for both processing and data transfer. There are two different types of 2 tier Templates: 2-Tier-a and 2-Tier-b. In a 2-Tier-a Template, the source node is used for data processing. Fig. 3 (a) shows the process, if 2-Tier-a architecture is used in a system. TRPS co-ordinates with the Task RA (1) and gives it a PBDT task and a resource pool (which is the set of all available nodes for this task). Task RA sends an acknowledgment signal back to TRPS (2). The Task Workflow Engine, deployed at Lower Level, L1, signals the source node to start the processing of data at the source node (3_1). The raw data file is processed at source node (3_2), and is delivered to each of the sink nodes (3_{31} to 3_{3k}). After the transfer of processed data is completed, each of the k sink nodes sends an acknowledgment to the Task Workflow Engine to indicate that the processed file have reached the sink nodes(shown by (4_1) to (4_k)). Once all the k sink nodes have sent completion signals to RA, RA sends the signal to TRPS to indicate that the task has been completed (5).

2-Tier-b Architectural Template is similar to 2-Tier-a (shown in Fig. 3 (b)). The important difference is that instead of using the source node, the data processing job is done at each of the sink nodes (3_{31} to 3_{3k} in Fig. 3 (b)).



(a)



(b)

Fig. 3. (a) 2-Tier-a Architecture (b) 2-Tier-b Architecture

6.2 4-tier Architectural templates

In a 4-tier Architectural Template, the resource pool of the given task (representing the set of available resources for the given task selected by TRPS) is grouped in two domains a compute-farm and a data-farm. Both a compute-farm and a data-farm have a specific role (see Fig. 4). The role the compute-farm is to process the data. Once all the data is processed, it is combined at the Egress node. The role of the data-farm is to replicate this processed data at chosen nodes to optimize its transfer to the sink nodes. Initially, TRPS co-ordinates with the RA and gives it a PBDT task and a corresponding resource pool (1). After running the resource allocation algorithm, RA generates the workflow of the given task T_i which indicates that which of the nodes from the provided resource pool will be used. RA returns $\acute{\omega}_i$ back to TRPS indicating which of the resources are planned to be used for the execution of task T_i (2). The Task Workflow Engine initiates the process (3). Once processing of the data is completed at the compute-farm nodes, these partitions are transferred to a special node called Egress Node where they are combined to produce the required processed file. The Egress Node sends a signal to the Task Workflow Engine (4) to indicate the completion of this stage.

The responsibility of the Egress node is to make sure that all the partitions of the raw data file associated with T_i have been successfully processed. Even if a small portion of data gets missing or corrupted due to any unforeseen error, the resultant processed file formed by the combination of the constituent processed files may be become invalid. In practical environments catching such error at earlier stage is often desirable as the Task Workflow Engine can re-initiate the processing of faulty data partition only. If Egress node is not present, the system is not able to catch such errors at early stages and in case of an error in the processing of one of the partitions, the resultant processed file becomes invalid. In this case the only way to recover is to restart the whole process again from the scratch which would be considerable wastage of both time and resources.

From Egress, this processed data is transferred to the data nodes chosen by the algorithm in the workflow. From there it is delivered to each of the k sink nodes. Once the processed data is delivered to all sink nodes, Task Workflow Engine is notified (5_1 to 5_k) which, in turn, notifies the TRPS (6) to indicate the completion of the task. Note that in compute-farm partitions of raw data files are transferred. But in data-farm complete processed files (not partitions) are transferred and replicated. 3-tier Architectural Template (having a compute-farm, but no data-farm or Egress node) is not discussed in this paper. If data-farm is not required, 3-tier Architectural Template can be used instead of a 4-tier Template.

7. RA Algorithms

In this section ATSRA algorithms are described which enable RA to assign resources to T_i within the resource pool, $\acute{\Gamma}_i$, allocated to it by TRPS. ATSRA algorithms are based on Linear Programming which is a popular technique for solving optimization problems [12][13]. It models an optimization problem as a set of linear expressions composed of input parameters and output parameters. The LP solver starts by creating a problem instance of the model by assigning values to the input parameters[16][17]. The problem instance is then subjected to an objective function, which is also required to be a linear expression. The values of the output variables, which collectively represent the optimal solution, are determined for the best value of the objective function. Based on this approach three algorithms are presented in this section which can be deployed at RA. Each of the ATSRA algorithms has the following two stages.

Stage-1: Selection of the most appropriate Architectural Template, AT_i for T_i.

Stage-2: Allocation of the resources for AT_i (if not done in stage 1.)

7.1 ATSRA$_{org}$ algorithm

A summary of the ATSRA$_{org}$ algorithm is presented. The algorithm is explained in more detail in the following section.

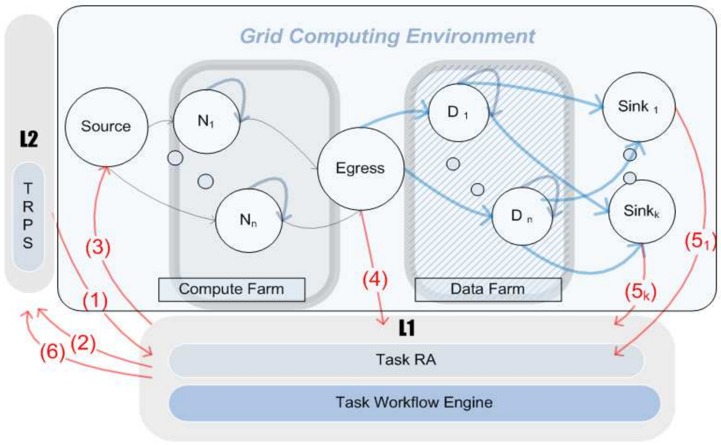


Fig. 4. 4-Tier Architecture

```
Summary of ATSRA_org algorithm
1 initialize
2 calculate $cost_{2\text{-Tier-a}}$ and $cost_{2\text{-Tier-b}}$
3 $cost_{min} = min(cost_{2\text{-Tier-a}}, cost_{2\text{-Tier-b}})$
4 calcuate $cost_{4\text{-Tier}}$
5 If  $cost_{4\text{-Tier}} < cost_{min}$
6     $cost_{min}=cost_{4\text{-tier}}$
7 choose Architectural Template associated with $cost_{min}$
8 Allocate resources for Architectural Template associated   with $cost_{min}$
```

First, cost associated with each of the Architectural Templates is calculated and the template having the minimum amount of total cost is chosen. The Architectural Template Selection phase starts with calculating the costs associated with the simplest of the templates. For PBDT tasks described in this paper, it starts with the 2-tier architectures.

If L is the size of the raw data file in MB, n_{src} is the source node, n_{sink_i} is the ith sink node, ξ is the processing factor associated with the given task T_i, Cp_{src} is the CPU processing cost per data unit at the source node, then total cost of performing the given PBDT task using 2-Tier-a architecture is

$$cost_{2\text{-Tier-a}} = L\,\{\textstyle\sum_{i=1}^{k} \xi d(n_{src}, n_{sink_i}) + Cp_{src}\,\} \tag{3}$$

If Cp_{Sink_k} is the cost of processing per data unit at the kth sink node then the total cost of performing a PBDT task using a 2-Tier-b architecture is given by

$$cost_{2\text{-Tier-b}}=L[\textstyle\sum_{i=1}^{k}\{d(n_{src}, sink_k) + \xi Cp_{Sink_k}\,\}] \tag{4}$$

For 4-tier cost calculations, the cost function is formulated as a mixed integer/linear programming (MILP) problem which is an LP problem with the additional restriction that certain variables must take integer values. MILP problems are generally h0arder to solve than LP problems [11]. If n_{src} is the source node, n_{sink_j} is the jth sink node, n_{egress} is the egress node and p is the number of partitions of the raw data file (as mentioned in its metadata); then for 4-tier Architectural Templates, the cost can be formulated as:

$$\text{cost}_{4\text{-tier}}=\min L\begin{bmatrix} \sum_{j=1}^{n}\frac{1}{p}\{Cp_j+d(n_{src},n_j)+\varepsilon d(n_j,n_{egress})\}x_i \\ +\ \varepsilon\sum_{i=1}^{n}d(n_{egress},n_i)y_i+\varepsilon\sum_{i=1}^{n}\sum_{j=1}^{k}d\left(n_i,n_{sink_j}\right)w_{ij}\end{bmatrix} \tag{5}$$

where x_i is a binary variable which is 1 if a particular node n_i is assigned to compute-farm and is 0 otherwise. Similarly y_i is a node assignment binary variable for the data-farm. It is 1, if a node is used for replication in the data-farm and is 0 otherwise. Variable w_{ij} is the *fraction* of the processed file that a sink gets from a particular node in data-farm. Note that we are considering $PBDT_{fixed\text{-}par}$ task with equal partitions, and each of these partitions at compute-farm has a length of L/p.

The feasibility of a particular assignment is determined by the following constraints.

1. $\sum_{i=1}^{n}w_{ij}=1 \qquad \forall\, j=1\ to\ k$
2. $\sum_{i=1}^{n}x_i=p$
3. $x_i \in \{0,1\}$
4. $y_i \in \{0,1\}$
5. $w_{ij}\le y_i \qquad i\in\Gamma, j\in S$
6. $w_{ij}\ge 0 \qquad i\in\Gamma, j\in S$

The first constraint specifies that the sum of all parts of the files being transferred from the data-farm to a particular sink node should add up to form the full length of the file. The second constraint specifies that the number of nodes used in the compute-farm should be equal to the number of partitions of the raw data file of the given task. The third and the fourth constraints ensure that both x_i and y_i are binary variables. The fifth constraint makes sure that the solution proposed by the algorithm has a non-zero value of w_{ij}, if and only if $y_i>0$. For example, consider a certain node n_3 and for a particular sink node s_7. w_{37} (that represents the portion of the total processed file that s_7 gets from n_3) should only have a non-zero value if $y_3=1$ (that is y_3 is being used as a node in data-farm). The last constraint prevents negative values for w_{ij}.

$$\text{Let}\quad \tilde{Cp_j}=Cp_j+d(n_{src},n_j)+\varepsilon d(n_j,n_{egress}) \tag{6}$$

Then $\tilde{Cp_j}$ represents the total cost of sending a unit data to a node in compute-farm, processing it and sending it to the egress node. From Equation (5) and Equation (6)

$$\text{cost}_{4\text{-tier}}=\min L\left[\frac{1}{p}\sum_{j=1}^{n}\tilde{Cp_j}x_i+\varepsilon\sum_{i=1}^{n}d(n_{egress},n_i)y_i+\ \varepsilon\sum_{i=1}^{n}\sum_{j=1}^{k}d(n_i,Sink_j)w_{ij}\right] \tag{7}$$

The input of the ATSRA algorithms are the computing cost vector [C_p] and the communication cost matrix [C_c. The output is the *solution matrix* which represents the values of solution variables, i.e.

$x_i, y_i \quad \forall\, i = 1$ to n
$w_{ij} \quad \forall\, i = 1$ to n, j=1 to k

It is important to note that there is nothing that prevents a node to be part of both compute-farm and data-farm. For example if the solution matrix has x_3=1 and y_3=1 and then n_3 is used both in data-farms and compute-farms. Once the costs calculated with all the Architectural Templates are calculated, the minimum of them is chosen. If $cost_{min}$= $cost_{4\text{-tier}}$, then resources are allocated according the values of the variables in solution matrix.

7.2 ATSRA$_{SSR}$ algorithm

For small number of nodes, ATSRA$_{org}$ algorithm performs well. But as the number of nodes increases the time taken by the algorithm to run becomes considerable. ATSRA$_{SSR}$ is proposed to improve performance for large number of nodes. It is based on finding a lower bound for the cost minimization problem formulated in Equation (7). The basic idea is to replace a "difficult" minimization problem by a simpler minimization problem whose optimal value is at least as small as $cost_{4\text{-tier}}$.

For the "relaxed" problem to have this property, there are two possibilities.

1. Enlarge the set of feasible solutions so that one optimizes. If the set of feasible solutions is represented by P, then it means to find P' such that $P \subseteq P'$.

OR

2. Replace the minimum objective function of Equation (7) by a function that has the same or a smaller value everywhere.

For the ATSRA$_{SSR}$, we have chosen the first approach. We formulate a new optimization problem called the relaxation of the original problem, using the same objective function but a larger feasible region P' that includes P as a subset. Because P' contains P, any solution which belongs to P, also belongs to P' as well. This relaxed cost is denoted by $cost_{4\text{-tier-relaxed}}$.. Tenlarge the set of feasible solutions, constraint relaxation technique is used. In ATSRA$_{SSR}$, the constraint relaxation technique is used at the Architectural Template selection stage only. Once an Architectural Template has been chosen, exact LP formulation is used for resource allocation. It is thus named as ATSRA Single Stage Relaxation (SSR) or ATSRA$_{SSR}$, as the constraint relaxation is applied only to first stage of the algorithm.

Summary of ATSRA $_{SSR}$ algorithm

```
1 initialize
2 calculate $cost_{2\text{-Tier-a}}$ and $cost_{2\text{-Tier-b}}$
3 $cost_{min} = min(cost_{2\text{-Tier-a}}, cost_{2\text{-Tier-b}})$
4 calcuate $cost_{4\text{-Tier-relaxed}}$
5 If $cost_{4\text{-Tier-relaxed}} < cost_{min} <$
6     goto step 9
7 else
      calculate $cost_{4\text{-Tier}}$
8 $cost_{min} = min(cost_{4\text{-Tier}}, cost_{min})$
9 choose Architectural Template associated with $cost_{min}$
10 Allocate resources for Architectural Template associated with $cost_{min}$
```

The ATSRA$_{SSR}$ starts by calculating the costs associated with 2- tier Architectural Templates 2a and 2b, using Equations (3) and (4). The minimum of these two is called as cost$_{min}$. For 4-tier Architectural Templates, instead of calculating exact cost$_{4\text{-tier}}$, cost$_{4\text{-tier-relaxed}}$ is calculated. For constraint relaxation, the fifth constraint (i.e. $w_{ij} \leq y_i$) is dropped.

7.3 ATSRA$_{BSR}$ algorithm

In ATSRA$_{BSR}$ algorithm we apply relaxation of the constraints at both Architectural Template Selection and resource allocation stages.

A summary of ATSRA$_{BSR}$ is as follows:

```
Summary of ATSRA_BSR algorithm
1 initialize
2 calculate $cost_{2\text{-Tier-a}}$ and $cost_{2\text{-Tier-b}}$
3 $cost_{min} = min(cost_{2\text{-Tier-a}}, cost_{2\text{-Tier-b}})$
4 calcuate $cost_{4\text{-Tier-relaxed}}$
5 If $cost_{4\text{-Tier-relaxed}} < cost_{min}$
6     $cost_{min}=cost_{4\text{-tier-relaxed}}$
7 choose Architectural Template associated with $cost_{min}$
8 Allocate resources for Architectural Template associated with $cost_{min}$
```

The important thing to note is that in ATSRA$_{BSR}$, the constraint relaxation technique is used at both the Architectural Template selection stage and the relaxed solution matrix is used for actual resource allocation. For relaxation, constraints 3 and 4 are replaced by

3′ $0 \leq x_i \leq 1$
4′ $0 \leq y_i \leq 1$

Note that the constraint relaxed in ATSRA$_{SSR}$ produces an invalid solution matrix. By dropping fifth constraint (i.e. $w_{ij} \leq y_i$), the variable w_{ij} can be assigned a non-zero value even if the corresponding data-farm node y_i is not assigned. Thus the resultant solution matrix cannot be used in resource allocation. But in ATSRA$_{SSR}$, we are using this relaxation only for the selection of Architectural Template and if 4-tier is chosen then the exact LP formulation is used for actual resource allocation. For ATSRA$_{BSR}$, we have chosen such constraints for relaxation that do not produce an invalid solution matrix. Thus the same resultant solution matrix is used for resources allocation as well.

Note that as we move from ATSRA$_{org}$ to ATSRA$_{SSR}$ and then to ATSRA$_{BSR}$, following are some of the considerations.

1. Time complexity of algorithm is reduced.
2. Imprecision in Resource Allocation increases.

The decrease in algorithm running time is the benefit of using this relaxation

8. Results experimental

This paper uses the following performance metrics.

Makespan total ($t_{ms\text{-}total}$): The time in seconds required for completing all the tasks in the given bag-of-tasks, T.

Makespan non-scheduling ($t_{ms\text{-}nonSch}$): The time in seconds required for completing all the tasks in T, *excluding* the time taken by scheduling algorithm.

Scheduling algorithm running time (t_{sch}): Total time in seconds taken by the scheduling algorithm to schedule all the given resources.

Total cost (t_{cost}). Sum of the time in seconds for which all the resources are utilized in performing the bag-of-tasks T.

To analyze the performance of the proposed RA algorithms, a simulation based investigation is performed. Various performance metrics described earlier were captured at the end of each simulation. Each experiment was repeated enough number of times to produce a confidence interval of ±5% for each performance metric at a confidence level of 95%. The workload chosen for these experiments is a bag-of-tasks consisting of 32 $PBDT_{fixed\text{-}par}$ tasks. Each of these tasks models the encoding of a raw multimedia file which is to be processed and delivered to a set of sink nodes. The choice of the raw data file is based on a typical animation movie described in [2]. The size of the raw data files of each of the tasks in the given bag-of-tasks is an important workload parameter. A detailed study of the characteristics of similar real-world tasks was carried out. The true representative probability distribution of the sizes of the raw or unprocessed data files used in similar tasks has been a subject of discussion over the years in the research community. Researchers seem to be split over characterizing it either with a Pareto or with Log-normal distribution. After careful analysis the Pareto distribution seems to be a better representative of PBDT multimedia workloads and is thus used. Another important parameter for the workload is the value of p, which is the number of partitions in which raw data files can be divided. This value is included in the metadata of each of the raw data file of the given tasks. The value of p depends on the structure of the raw data file and the type of processing required for it. For example, if a raw data file of multimedia animation contains 20 sub-sequences, each of which has to be processed as a single partition, then this task has a p of 20. The number of partitions (referred to as sub-sequences in the description of the movie rendering project presented in [1]) for each raw file varies from 1 to 30. We have used a uniform distribution [1-30] for modeling the number of partitions in each raw multimedia file. The mean of the raw data files is fixed at 650MB.

For performance analysis of the proposed algorithms total number of nodes of the Grid system is increased. Number of Grid nodes is directly related to the time-complexity of the deployed algorithm. All other parameters related to workload and system conditions are kept constant.

Fig. 5 shows the performance of three RA algorithms ($ATSRA_{org}$, $ATSRA_{SSR}$ and $ATSRA_{BSR}$) with SRP_{sp} deployed at the TRSP. Fig. 5(a) shows the time taken by each of the algorithm to run. It can be that for small number of nodes, there is not much difference in the scheduling time taken be these three algorithms. The time taken by the $ATSRA_{org}$ algorithm rises sharply as number of nodes is increased more than 32. It can be observed that for $ATSRA_{BSR}$, t_{sch} does not rise sharply. Fig. 6(b) shows the value of Makespan non-scheduling ($t_{ms\text{-}nonSch}$) for each of the proposed algorithms as the number of nodes is increased. It is clear that $ATSRA_{org}$ has the lowest value of $t_{ms\text{-}nonSch}$ for all values of n. This is expected as by using all

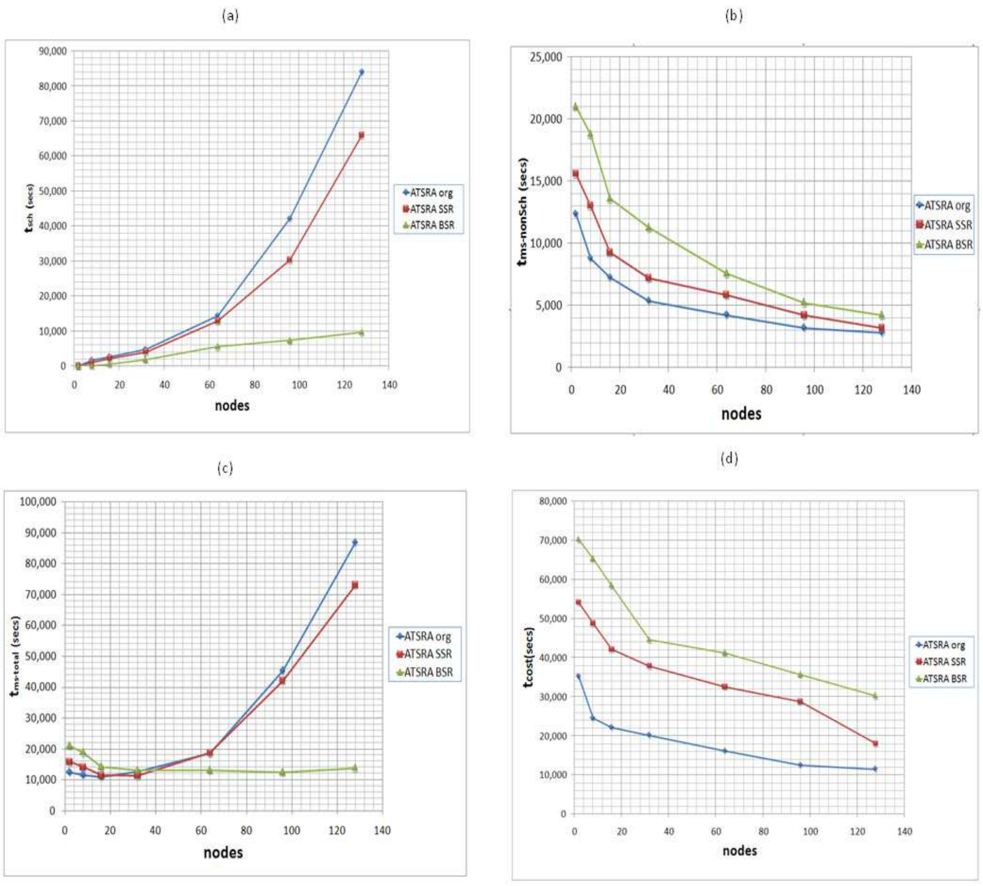


Fig. 5. Performance with SRP_{sp} chosen at TRPS (a) Scheduling Algorithm Running time (b) Makespan non-Scheduling time (c) Makespan Total (d) Total Cost

constraints associated with the LP formulation, we are allocating the resource with the highest precision and it this allocation is expected to be efficient. As constraint relaxation is applied at stage one of the algorithms in $ATSRA_{SSR}$, $t_{ms\text{-}nonSch}$ increases. It further increases for $ATSRA_{BSR}$ in which constraint relaxation is applied at both stage of the algorithm. Fig. 6(d) shows the total cost (t_{cost}) for each of the three algorithms. $ATSRA_{org}$ has the lowest t_{cost} as we are allocating the resources with highest precision. This is followed by $ATSRA_{SSR}$ and $ATSRA_{BSR}$. The overall makespan time, $t_{ms\text{-}total}$ shown in Fig. 5(c) includes both the scheduling time and the execution time for the bag-of-tasks. It captures the tradeoff between $t_{ms\text{-}noSch}$ and t_{sch} presented in Fig. 5(b) and 5(a) respectively. For a very small number of nodes the scheduling overhead for the $ATSRA_{org}$ is small and $t_{ms\text{-}nonsch}$ is the lowest and as a result the best $t_{ms\text{-}total}$ is achieved. For a large number of nodes, the scheduling overhead for $ATSRA_{org}$ is very high and the benefit of using a better resource allocation is offset by the overhead and its performance deteriorates. $ATSRA_{BSR}$ that exhibits the smallest scheduling overhead for a large number of nodes (see Fig. 5(a)) demonstrates the best $t_{ms\text{-}total}$ (see Fig.

5(c)). It is interesting to see that $ATSRA_{SSR}$ produces the best $t_{ms-total}$ for a range of intermediate values of the number of Grid nodes. The accuracy of resource allocation for $ATSRA_{SSR}$ lies between that achieved with $ATSRA_{org}$ and $ATSRA_{BSR}$. For a small number of nodes, t_{sch} of $ATSRA_{SSR}$ is comparable to that of $ATSRA_{org}$; whereas the $t_{ms-nonSch}$ achieved by $ATSRA_{SSR}$ is inferior to that achieved by $ATSRA_{org}$. Thus if the number of nodes is small, $ATSRA_{SSR}$ is inferior to that of $ATSRA_{org}$.

For a large number of nodes, although $ATSRA_{SSR}$ gives rise to a lower scheduling overhead than $ATSRA_{BSR}$, the advantage is offset by the much lower execution time produced by $ATSRA_{BSR}$. The net effect is that $t_{ms-total}$ achieved by $ATSRA_{SSR}$ is inferior to that of $ATSRA_{BSR}$ for a large number of nodes.

Fig. 6 shows the performance of ATSRA algorithms when SRP_{sp} +BF is deployed at TRPS. As in the case of Fig. 5(c) the best $t_{ms-total}$ is achieved by $ATSRA_{BSR}$ for larger numbers of nodes; whereas $ATSRA_{org}$ demonstrates the best performance for a lower number of nodes. $ATSRA_{SSR}$ demsonstrates a slightly higher $t_{ms-total}$ than $ATSRA_{org}$ when the number of Grid nodes is small. Although the total makespan achieved by it is better than $ATSRA_{org}$ at higher number of nodes, it is higher than that achieved by $ATSRA_{BSR}$. The relative performances of the three algorithms captured in Fig. 6(a) , Fig. 6(b) and Fig. 6(d) are the same as those displayed in Fig. 5(a), Fig. 5(b) and Fig. 5(d) respectively. $ATSRA_{org}$ demonstrates the best in $t_{ms-nonSch}$ and t_{cost} followed by $ATSRA_{SSR}$ and $ATSRA_{BSR}$; whereas the smallest scheduling overhead is achieved with $ATSRA_{BSR}$ and $ATSRA_{org}$ demonstrates the highest scheduling overhead. The rationale for such a behavior has been provided in the discussion presented earlier for Fig. 5(a) Fig. 5(b) and Fig. 5(d). Note that although the shapes of the graphs in Fig. 5(a) and Fig. 6(a) are similar, the value of t_{sh} for a given number of nodes in Fig 6(a) is higher than the value of t_{sh} for the same number of nodes in Fig. 5(a). This is because in SRP_{sp} +BF backfilling is used which increases scheduling overheads. While the relative performance of $ATSRA_{org}$, $ATSRA_{SSR}$ and $ATSRA_{BSR}$ remains almost the same, this additional scheduling overhead has shifted the graphs upwards in Fig. 6(a) as compared to Fig. 5(a).

For $ATSRA_{org}$ and $ATSRA_{SSR}$ algorithms and any given number of nodes, the $t_{ms-nonSch}$ achieved with SRPsp +BF is observed to be smaller than that achieved SRP_{sp} (see Fig. 5 (b) and Fig. 6(b). This demonstrates the effectiveness of using backfilling that can increase the concurrency of task execution. Except for the case in which the number of Grid nodes is 128, a similar behavior is observed with $ATSRA_{BSR}$.

Comparing $t_{ms-total}$ achieved with SRP_{sp} (Fig. 5(c)) and SRP_{sp} +BF (Fig. 6(c)), we observe that for any given ATSRA algorithm, the total makespan achieved by SRPsp +BF is superior to that achieved by SRPsp when the number of nodes is small. For higher number of nodes, SRP_{sp} +BF demonstrates an inferior performance. This becauseat smaller number of nodes concurrent execution of tasks may be severely limited with SRP_{sp} because many tasks may not be able to get all their resources at the same time. With the use of backfilling this problem is alleviated as RA is run for each waiting task with the set of unused resources as the resource pool. However, this problem with task concurrency is not that severe at higher number of nodes. Thus, SRP_{sp} +BF that re-runs RA multiple times and incurs a higher scheduling overhead demonstrates an inferior performance as the potential performance benefit due to backfilling is offset by the overhead.

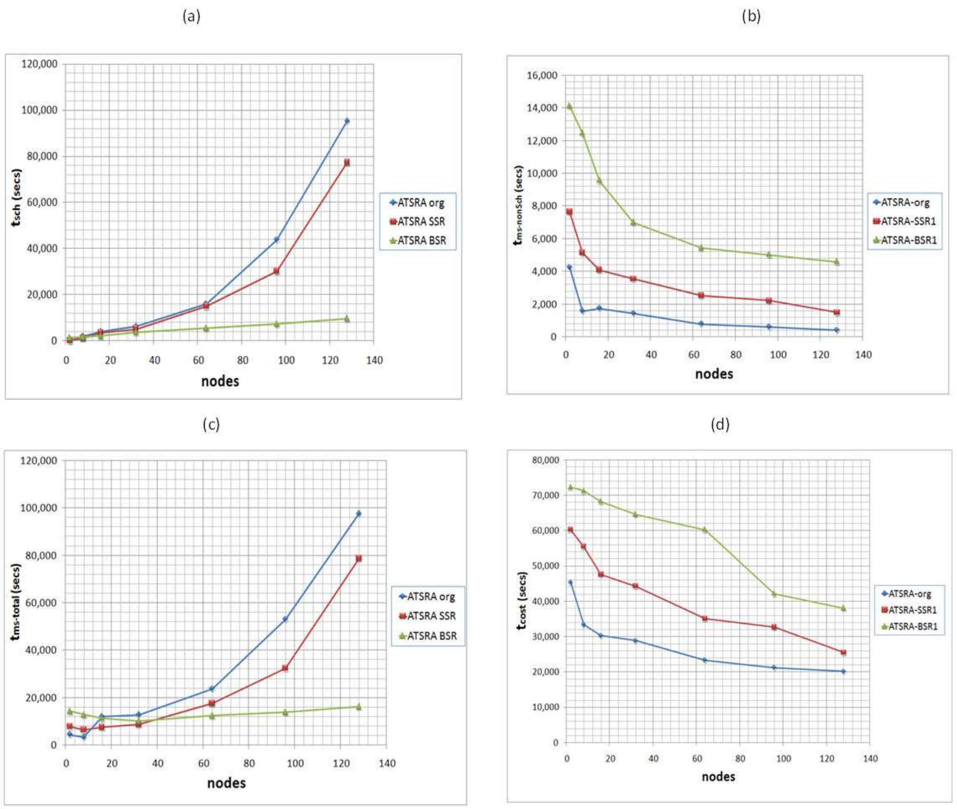


Fig. 6. Performance with SRP_{sp}+BF chosen at TRPS (a) Scheduling Algorithm Running time (b) Makespan non-Scheduling time (c) Makespan Total (d) Total Cost

9. Summary and conclusion

In this chapter, by using BiLeG an allocation-plan is devised which reflects the overall resource allocation strategy comprising two parts; a policy used at the higher decision making module, TRPS, which has the responsibility to select a resource-pool for each of the tasks; and a resource allocation algorithm used at the lower decision making module, RA, which actually assigns resources from the resource-pool selected by TRPS for a particular PBDT task. Three RA algorithms and six TRPS policies have been proposed in this chapter forming different allocation-plans. The suitability of various allocation-plans under different sets of system and workload parameters has been explored.

Detailed study of the various trade-offs, implicit in the use of different allocation-plans, is the focal points of this chapter. The most suitable allocation-plan not only depends on various workload and system parameters, it also depends on the user requirements and the hardware available. It can be seen that from the performance perspective various trade-offs exist among different allocation-plans and understanding these trade-offs in depth is the focus of the experiments conducted in this chapter.

For the choice of an appropriate allocation-plan, two of the important considerations that came out of these experimental results are the size of the Grid and the performance metric chosen for optimization. Generally, from the results obtained from the experiments conducted in chapter, it can be concluded that if an allocation-plan tries to minimize one of the performance metrics, it tends to yield higher values of the other performance metrics. For example, $<SRP_{sp},ATSRA_{org}>$ always gives the lowest value of t_{cost} but it also yields one of the highest values for $t_{ms\text{-}WOH}$, especially for a large number of nodes. At RA, the trade-offs associated with reducing the accuracy of the ATSRA algorithm by relaxing some of the constraints in the LP formulation have been studied. The combination of the proposed RA algorithms and TRPS policies gives rise to various allocation-plans. These allocation-plans can be used under a wide variety of system and workload parameters to maximize the use of available resources according to a pre-determined optimization objective.

Although the research summarized in this chapter has focused primarily on the Grid systems, the proposed BiLeG architecture can also be used in a Cloud Computing environment. Cloud Computing environments are often classified as public and private Cloud environments [3]. The private Cloud environment is better suited for the BiLeG architecture; as a private Cloud environment uses a dedicated computing infrastructure that provides hosted services to a limited number of users behind a firewall and can, thus, more easily incorporate mechanisms to accurately predict the computing and communication costs.

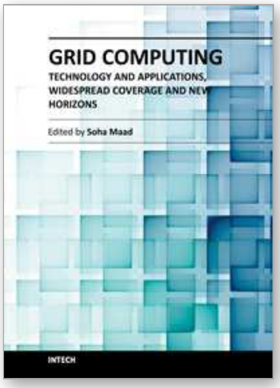
The algorithms presented in this chapter are based on a dedicated resource environment. To adapt the BiLeG architecture to shared environments, more research is required. For example, in order to use it in a shared resource environment, mechanisms to accurately predict the unit communication and processing times are needed to be incorporated in the BiLeG architecture. Also, in a shared environment, investigating the impact of various types of communication models, such as many-to-one and one-to-many forms, an important direction for the future research.

10. References

[1] http://enterprise.amd.com/Downloads/Industry/Multimedia
[2] http://www.gridtoday.com/grid/638845.html.
[3] Abbas A. Grid Computing: A Practical Guide to Technology and Applications, Charles River Media , 2004.
[4] Ahmad I. and Majumdar S. Policies for Efficient Allocation of Grid Resources using a Bi-level Decision-making Architecture of "Processable" Bulk Data. Department of Systems and Computer Engineering, Carleton University, 2007.
[5] Ahmad I. and Majumdar S. An adaptive high performance architecture for "processable" bulk data transfers on a Grid. In 2nd International Conference on Broadband Networks (Broadnets). (3-7 Oct. 2005). IEEE, Boston, MA, USA, 2005, 1482-91.
[6] Ahmad I. and Majumdar S. Efficient Allocation of Grid Resources Using a Bi-level Decision-Making Architecture for "Processable" Bulk Data. On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, (2007), 1313-1321.
[7] Allcock B., Chervenak A., Foster I., Kesselman C. and Livny M. Data Grid tools: enabling science on big distributed data. Journal of Physics: Conference Series, 16, 1 (2005), 571-5.

[8] Bunn J. and Newman H. Data-intensive Grids for high energy physics. In Berman G. and Hey E. eds.Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons, Inc., New York, 2003.
[9] Berman F., Casanova H., Chien A., Cooper K., Dail H., Dasgupta A., Deng W., Dongarra J., Johnsson L., Kennedy K., Koelbel C., Liu B., Liu X., Mandal A., Marin G., Mazina M., Mellor-Crummey J., Mendes C., Olugbile A., Patel M., Reed D., Shi Z., Sievert O., Xia H. and YarKhan A. New Grid scheduling and rescheduling methods in the GrADS project. International Journal of Parallel Programming, 33, 2-3 (06 2005), 209-29.
[10] Candler W. and Townsley R. A linear two-level programming problem, Computers & Operations Research, 9, 1 (1982), 59-76.
[11] Chvatal V. Linear Programming. W.H. Freeman and Company Press, 1980.
[12] Devpura A. Scheduling Parallel and Single Batch Machines to Minimize Total Weighted Tardiness. Ph.D. Dissertation, Computer Science Department, Arizona State University, 2003.
[13] Dimitris Bertsimas, John N. Tsitsiklis. Introduction to Linear Programming. Athena Scientific, Belmont, Massachusetts, 1998.
[14] Downey A. B. Lognormal and Pareto distributions in the Internet. Comput. Commun., 28, 7 (05/02 2005), 790-801.
[15] Foster I. and Kesselman C. The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, USA, 1999.
[16] Gzara F. Large scale integer programming: A novel solution method and application. McGill University, 2004.
[17] Jeong S., Chan-Hyun Youn and Hyoug-Jun Kim. Optimal file replication scheme (CO-RLS) for data Grids. In Anonymous 6th International Conference on Advanced Communication Technology. (9-11 Feb. 2004). IEEE, Phoenix Park, South Korea, 2004, 1055-9.
[18] Foster I., Kesselman C., Lee C., Lindell B., Nahrstedt K. and Roy A. A distributed resource management architecture that supports advance reservations and co-allocation. In Anonymous Proceedings of IWQoS'99 - Seventh International Workshop on Quality of Service. (31 May-4 June 1999). IEEE, London, UK, 1999, 27-36.
[19] M. Elayeb. Efficient Data Scheduling For Real-Time Large Scale Data Intensive Distributed Applications. (Masters Dissertation, The Ohio State University).
[20] Mathur K. and Puri M. C. A bilevel bottleneck programming problem. European Journal of Operational Research, 86, 2 (10/19 1995), 337-344.
[21] Sander V., Adamson W. A., Foster I. and Roy A. End-to-end provision of policy information for network QoS. In 10th IEEE International Symposium on High Performance Distributed Computing. (7-9 Aug. 2001). IEEE Comput. Soc, San Francisco, CA, USA, 2001, 115-26.
[22] Sundaram B. and Chapman B. M. XML-based policy engine framework for usage policy Allocation in Grids. In Proceedings. (18 Nov. 2002). Springer-Verlag, Baltimore, MD, USA, 2002, 194-8.
[23] Vazhkudai S. Enabling the co-allocation of Grid data transfers. In Fourth International Workshop on Grid Computing. (17 Nov. 2003). IEEE Comput. Soc, Phoenix, AZ, USA, 2003, 44-51.

[24] Venugopal S. Scheduling Distributed Data-Intensive Applications on Global Grids. Ph.D. Dissertation, Department of Computer Science and Software Engineering, The University of Melbourne, 2006.
[25] Verma D., Sahu S., Calo S., Beigi M. and Chang I. A policy service for Grid computing. In Proceedings (18 Nov. 2002). Springer-Verlag, Baltimore, MD, USA, 2002, 243-55.
[26] Wu J., Savoie M., Campbell S., Zhang H., Bachmann G. V. and St Arnaud B. Customer-managed end-to-end lightpath provisioning. International Journal of Network Allocation, 15, 5 (2005), 349-62.
[27] Yang K., Galis A. and Todd C. Policy-based active Grid Allocation architecture. In Towards Network Superiority. (27-30 Aug. 2002). IEEE, Singapore, 2002, 243-8.


GRID COMPUTING
TECHNOLOGY AND APPLICATIONS, WIDESPREAD COVERAGE AND NEW HORIZONS
Edited by Soha Maad
INTECH

Grid Computing - Technology and Applications, Widespread Coverage and New Horizons
Edited by Dr. Soha Maad

ISBN 978-953-51-0604-3
Hard cover, 354 pages
Publisher InTech
Published online 16, May, 2012
Published in print edition May, 2012

Grid research, rooted in distributed and high performance computing, started in mid-to-late 1990s. Soon afterwards, national and international research and development authorities realized the importance of the Grid and gave it a primary position on their research and development agenda. The Grid evolved from tackling data and compute-intensive problems, to addressing global-scale scientific projects, connecting businesses across the supply chain, and becoming a World Wide Grid integrated in our daily routine activities. This book tells the story of great potential, continued strength, and widespread international penetration of Grid computing. It overviews latest advances in the field and traces the evolution of selected Grid applications. The book highlights the international widespread coverage and unveils the future potential of the Grid.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Imran Ahmad and Shikharesh Majumdar (2012). Resource Management for Data Intensive Tasks on Grids, Grid Computing - Technology and Applications, Widespread Coverage and New Horizons, Dr. Soha Maad (Ed.), ISBN: 978-953-51-0604-3, InTech, Available from: http://www.intechopen.com/books/grid-computing-technology-and-applications-widespread-coverage-and-new-horizons/resource-management-for-data-intensive-tasks-on-grids