International Journal of Grid and Distributed Computing
Vol.7, No.1 (2014), pp.149-168
http://dx.doi.org/10.14257/ijgdc.2014.7.1.14

A Comprehensive Performance Tuning Scheduling Framework for Computational Desktop Grid

K. Hemant Kumar Reddy[1], Diptendu Sinha Roy[2] and Manas Ranjan Patra[3]

[1,2]*National Institute of Science & Technology, Berhampur*
[3]*Dept. of Computer Science, Berhampur University, Berhampur*
[1]*khemant.reddy@gmail.com*, [2]*diptendu.sr@gmail.com*, [3]*mrpatro123@gmail.com*

Abstract

As computers are pervading in all aspects of life with advancement of technology, a growing demand is being felt for low-cost, huge computational need. Grid computing paradigm provides an attractive alternative where enormous computational and data processing and management power can be generated at a much cheaper budget by means of large scale sharing of resources. However, the efficacy of such systems largely depends on the efficacy of scheduling policies employed. In the past few years, myriad, novel scheduling algorithms have been proposed. Besides, for long running and similar types of applications the performance of grid systems can be improved further if the scheduled jobs can be tuned at run time. It is a very important aspect for enhancing performance, considering the drastic fluctuation of resources availability, variations in communications bandwidth, fluctuations in job submission frequency etc that characterize a grid scenario. In this paper, the design and implementation of a Comprehensive Performance Tuning Framework (CPTF) is reported that initially schedules jobs to resources, mines all such job-to-resource mapping information and thereafter tunes certain parameters for subsequent scheduling of submitted jobs. CPTF aims to minimize the overall throughput of the system instead of minimizing single job execution time. Experiments result shows the efficacy of the proposed framework under varying load conditions.

Keywords: *Grid Scheduling, Performance Tuning, Adaptive Scheduling, CPT F, GridGain*

1. Introduction

Grid computing allows the users the facility of large scale computational and data handling capabilities by employing large-scale sharing of resources. The true worth of grid computing lies in the fact that it caters enormous computational power for users at a cost drastically less than conventional supercomputing infrastructures [1]. But grid is a heterogeneous system, contrary to traditional clusters or supercomputers. In order to effectively use the potential of grids, jobs have to be scheduled proficiently thus calling for designing effective scheduling algorithms. The job of a scheduling algorithm is to assign jobs to resources in a grid, which acts like a virtual supercomputer [2] having its resources distributed across a network. Thus unlike scheduling parallel systems like supercomputers that have reliable communication, scheduling in grid infrastructures is much more complicated, in the sense that many more conditions need to be considered. Network status is the form of available network bandwidth, resources status in the form of available computing capacity, available heap memory and so forth are a few such conditions that need to be considered for effective scheduling.

ISSN: 2005-4262 IJGDC

Additionally the afore mentioned availability of network status and resource status can vary drastically during the execution of jobs, thus complicating further the general tasks scheduling problem, which is an NP-complete problem [3]. Unfortunately, due to possible fluctuations in resources availability, the scheduling policy needs to be adjusted from time to time for performance enhancement. Thus, in order to extract better performance from grids, effective job scheduling should be complimented with a performance monitoring and tuning mechanism. This need becomes more prominent considering that applications that run on such grid may include independent jobs that can be split to sub jobs at run time.

Tuning the performance of applications is a well studied field for parallel systems where the underlying architecture is known along with the interconnection pattern. The challenge of developers lies in effective utilization of application characteristics on specific architecture that leads to efficient deployment. But this process is highly manual demanding expertise to identify performance bottleneck, identify the cause from performance data by correlating run time behavior with program characteristics; formulating hypothesis and subsequently considering validation tests [4]. Such expertise to enable applications demands enormous expertise possessed by few engineers. Moreover applications enabling on highly specific architecture can be dealt by compiler level optimization [5, 6]. Grids are heterogeneous and dynamic in nature and thus optimizations tailor-made for target architectures are not practicable. Rather, a natural alternative for tuning applications on grid platform would be to tune applications based on good configuration, much analogous to unsupervised machine learning. Reference [7] presents 'Active Harmony', as automated approach for automatic performance tuning on supercomputers. The unique contribution of this paper is to formulate heuristics for tuning performance of grid application at execution time based on job types, job size and grid size. Owing to dynamic change in resources status, unreliability of network, fluctuations in job submission frequencies, a particular scheduling algorithm might produce markedly varying performances-thus hinting the need of run-time tuning of applications on grids.

2. Related Work

There have been some researches covering performance tuning of applications on grid test beds. The Grads project [8] integrated application monitoring and adaptive control services within their framework. They employed the Globus Toolkit [9] for middleware services to leverage effective scheduling policies for certain computation-intensive numerical solutions and deployed rescheduling or redistribution of resources once quality of service falls beyond acceptable range. Huedo *et. al.*, [10] presents a Globus based framework, called Grid way, that schedules jobs on a dynamic grid in a "adaptive and submit and forget" fashion. Sarbani Roy *et. al.*, [11, 12] presented a framework (PRAGMA) where task migration and rescheduling of batch of jobs, both for super computer and clusters, were implemented by rescheduling the job on a new host either with the help of check pointing files or by alternatively rescheduling the job from beginning if check pointing fails. The adaptive execution framework was implemented by means of a multi-agent system that employed Jini and deployed job migration using mobile agents. A multi-layer resource reconfiguration framework for performance enhancement of user programs running on grid was proposed and implemented by Chen *et. al.*, [13]. This framework implements different resource configuration applications for different workloads of resources. The aforementioned contributions deal more or less with typical computational jobs with very restricted variety. These frameworks needed adequate performance modeling. [14, 15] presented

self-adaptive grid scheduling without the need to use performance model. In this paper, both dependent as well as independent jobs (decomposable to finer sub jobs for parallel scheduling) are considered under varying resources availability conditions to embark upon a performance tuning framework, henceforth which is referred to as the Comprehensive Performance Tuning Framework (CPTF). The philosophy of CPTF is based on the following capabilities (i) a job decomposition analyzer that adaptively selects the scheduling criteria and decomposes independent jobs for subsequent mapping to grid nodes. (ii) a dynamic decision policy to decide what fraction of decomposed subtasks to be processed locally taking into account network availabilities (L-R Strategy) and (iii) a post scheduling monitoring mechanism that identifies performance degradation beyond certain thresholds and (iv) subsequent rescheduling scheme.

In order to investigate the capabilities of the CPTF, a test bed of desktop computers has been deployed to set up a grid using GridGain 2.0 [16]. A wide variety of jobs are run on this grid test bed and the experiments are repeated multiple times with varying network bandwidth as well as local computational load conditions in order to assess the tuning capability of the proposed framework. The work presented in this paper builds upon the adaptive scheduling framework proposed and implemented in an earlier article [17, 21], by embedding capabilities like rescheduling (migrating) jobs (or sub-jobs) by means of appropriate tuning parameters. The detailed design of the CPTF has been presented in Section 3. The results presented in this paper show the efficacy of the CPTF to tune performance for achieving better performance under varied network bandwidth as well as varied computation load among grid nodes. Thus the name Comprehensive Performance Tuning Framework (CPTF) has been chosen.

The rest of this paper is organized as follows: Section 2 presents a brief outline of scheduling in grid system. Section 3 introduces the Comprehensive Performance Tuning Framework (CPTF) along with a detailed discussion of its architecture, execution life cycle of jobs in the framework and related information. Sections 4 and 5 give an in-depth account of the two major components of the CPTF, namely the adaptive scheduling model and the performance tuning model respectively. Results are presented and subsequently analyzed in Section 6. Conclusions are presented in Section 7.

3. Desktop Grid Scheduling: A Brief Review

The concept of grid computing roots from the vision of utilizing geographically distributed desktop computers linked through the Internet in a Grid-like fashion, much like the electric power systems' sharing of power among geographically distributed generation and load points through grid stations. In essence, the vision of grid computing was to make virtual supercomputing a reality by mass-scale sharing of resources via interconnection. For a number of reasons, like heterogeneity of resources, unpredictability of their availability patterns, unreliability of the communication and so forth, scheduling jobs in a grid is much complicated a decision than in traditional systems, including supercomputing infrastructures [1, 2]. Scheduling of jobs can be simply done by assigning the incoming tasks to the available compatible resources for simplest of systems [3]. However, use of more advanced and sophisticated schedulers can greatly affect the performance of grid scheduling. Grid schedulers are generally expected to react to the dynamics of the Grid system by keeping track of its fluctuating pool of components that may join or leave the grid at any instant. Further, schedulers can be managed in a hierarchical form or in a distributed manner to deal with the large scale of Grids. Reference [3] surveys some of the most important and useful

computational models employed for grid scheduling and focus on the design of efficient Grid schedulers using several heuristic and meta-heuristic methods. It also analyzes why heuristic and meta-heuristic schemes are good alternatives to conventional scheduling strategies and what make them suitable for Grid scheduling.

As grid technology is receiving widespread acceptance in many domains, a numerous variety of grids have come into existence. Desktop Grid has emerged in institutions having a large number of mostly underutilized desktop PCs. Desktop Grids are engaged in effectively utilizing the idle cycles of the underutilized desktop PCs. They are characterized by heterogeneous capabilities, failures, volatility, and lack of trust as it is based on desktop computers from the view of internet. A desktop grid computing environment mainly consists of clients, worker nodes, and a server. A client acts as a parallel job submission portal. A worker node is a resource provider that donates its computing resources. The server acts as the central manager for controlling the submitted jobs and resources. A client submits a parallel job to a server. A job is divided into sub-jobs, called tasks. The server allocates tasks to worker nodes using a scheduling mechanism. Each worker node executes its task, while continuously requesting data from its server. When each worker node subsequently finishes its task, it returns the result of the task to the server. Finally, when the server collects all results of tasks from worker nodes, it returns the final result of the job back to the client.

Scheduling in desktop grids is different from that of cluster grids, since a desktop grid may vary widely depending upon the type of resources, dedication, trust, failure modes, applications, and so forth. Grid scheduling is the process of assigning jobs to the most appropriate resources. Scheduling may be performed in a centralized fashion or in a fully distributed way [18]. For the most part, desktop grid systems do not need any local scheduler unlike clustered grids since in this case the scheduling target is a single desktop computer, contrary to a site in conventional Grid [19]. In desktop grid, communication cost is very high as compared to cluster computers or super-computers. In hierarchical desktop grid environment, it is very difficult to classify grid resources into groups as compared to cluster computers or high performance systems. This is so because there is no single parameter that can effectively classify grid resources into groups and decide the next levels of scheduling. Finally, desktop grid scheduling is opportunistic. Desktop grid respects the autonomy of worker nodes (that is, worker nodes can freely participate in public execution). In this work, a test-bed model has been described that provides distributed scheduling, much like in peer to peer systems. For the purpose of demonstrating the efficacy of the proposed CPTF, a grid test bed has been deployed using GridGain. GridGain is a Java based middleware that provides the necessary tools and features to transparently leverage the low-level issues on behalf of the users, thereby providing a functional environment that captures the essence of grid systems. The job scheduling model for the aforesaid test-bed has been presented in the next section.

4. The Comprehensive Performance Tuning Frame Work (CPTF): A Brief Overview

The aim of CPTF is to reduce the overall completion time of the submission. In order to meet this goal, CPTF considers several parameters for effective scheduling of jobs. In most literatures [3, 10, 11, 12], computing power of each resource in a grid is taken into account. Reference [2] accounts for transmission power of each cluster in addition to computing power of grid of resources. CPTF takes into account computational power availability and heap memory availability during scheduling of jobs. It incorporates an adaptive scheme [17] for choosing appropriate parameters based on which effective

scheduling can be done. As mentioned in Section 1, tuning an application can be manually accomplished knowing the target configuration and capacities, but such endeavors seem inappropriate for grids. Thus CPTF attempts to keep track of suitable job-resource mapping by mining performance data from previous runs. As in most service oriented computational schemes, CPTF collects some service level agreement (SLA) related information of a job, like job type, job size and so on for effective mining of these information. In this paper four different jobs are employed for assessing the performance of the CPTF, Table 1 shows the SLA information to be provided by the user for each of these job types.

The CPTF consist of the several components, primary to which are the Adaptive Scheduling Model (ASM) and the Performance Tuning Model (PTM) as has been depicted by dotted lines in Figure 1. The ASM helps adaptively choose a parameter based on which to schedule tasks. Possible parameters are CPU availability, Heap memory availability or the combined mean availability of these two parameters and so forth. CPTF supports various models of operation. It can allow job submission in online mode as well as in batch mode. Jobs submitted can either be dependent jobs as well as independent jobs, those which can be decomposed and further to tasks simultaneously processed at different grid nodes just like in a Map-Reduce scenario. The following subsection provides a detailed description of the system framework. The desktop grid setup allows users to submit their jobs through any of the terminals, which acts as the grid portal. Appropriate authentication of users prompts the users to enter few SLA parameters as detailed in Table 1. The entered parameters are different for batch mode and online mode of operation.

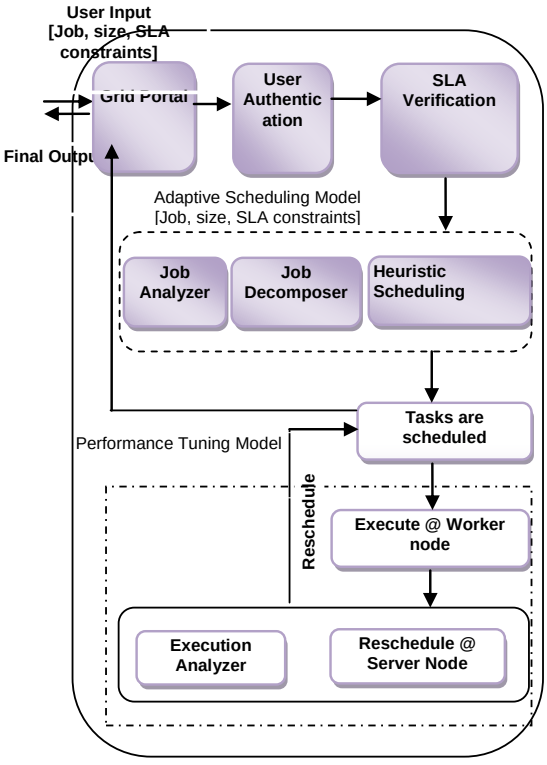

Figure 1. The CPTF Architecture

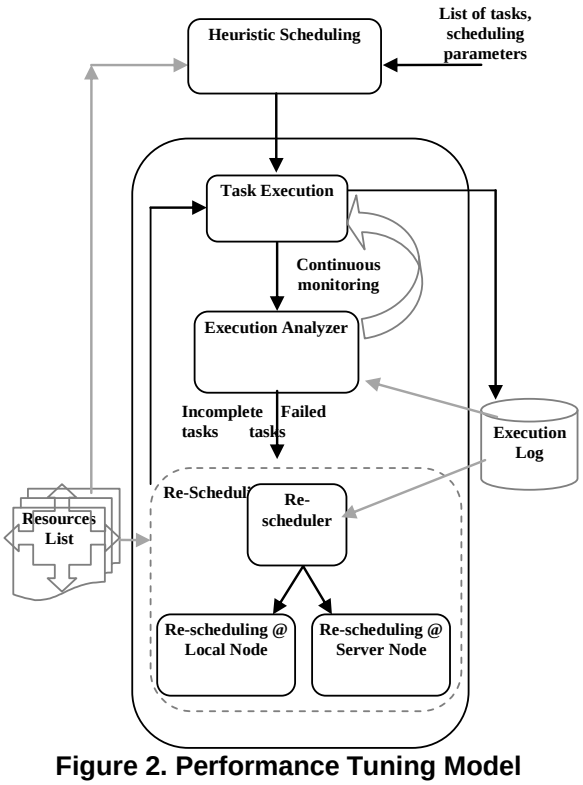

Figure 2. Performance Tuning Model Interactions

4.1. System Framework

This section presents the CPTF framework that has been deployed at the High Performance Computing (HPC) lab at National Institute of Science and Technology campus [17, 21] along with other associated computer laboratories within the campus. Jobs can be submitted through any of the desktop PCs, which act as the grid portal. After appropriate authentication and sub sequent SLA specifications, jobs are assigned to the scheduler. The scheduler is responsible for mapping the tasks to appropriate resources within the participating grid nodes after decomposition in case of independent jobs, whereas dependent jobs are directly assigned to appropriate resources within the participating grid nodes. For independent jobs that can be executed in parallel, the job decomposer is responsible for splitting a job to multiple sub jobs. This is done with the help of information provided by job analyzer. Keeping in view the compute / communication overhead for executing a job at local or remote machine, an L-R Factorization scheme decides upon a proper ratio of mapping jobs to local and remote machines. Based on data from execution logs, the best parameter for scheduling is chosen for the present set of submitted jobs. The job scheduler model is thus adaptive, because the parameters based on which jobs are scheduled vary at run time. CPTF provides the facility to constantly monitor all jobs and the execution history is mined. An execution analyzer is designed to identify glitches in performance and job can be rescheduled if needed. The following subsection provides a brief account of the steps that submitted jobs in the CPTF has to undergo.

4.2. Job Execution Life Cycle

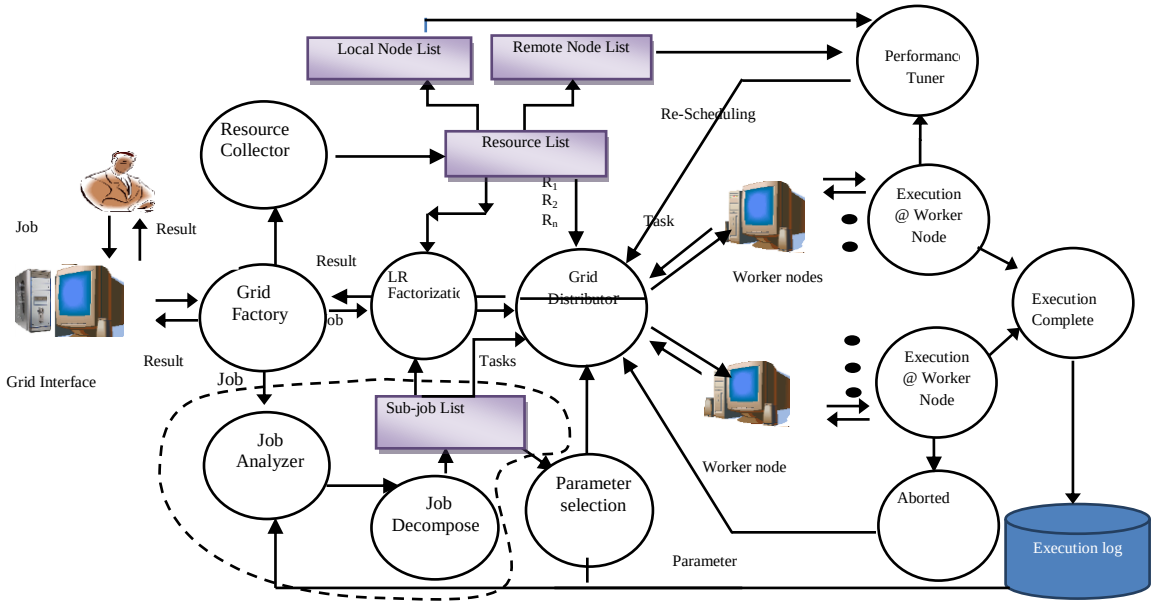

Figure 2. Job Execution Life Cycle for Independent Jobs

Any user submitted job in the CPTF framework undergoes several states before it can be scheduled to a worker node and can be subsequently get executed. All these states taken together constitute a job's life cycle. Figure 3 shows the different states through which an independent job is scheduled on this framework. The users submit their jobs to the grid system through a portal, which eventually forwards all such submitted jobs

to grid distributor. The distributor class is one of the essential components which is continuously updated by the latest status information of all constituent grid nodes. Based on the availability status information and previous execution history, jobs are scheduled to grid nodes adaptively. The execution history is maintained as in long for scheduling of future jobs. Dotted portion of the figure 3 shows the job life cycle for dependent jobs, the essential distinction being the absence of job analyzer and decomposer

5. The Adaptive Scheduling Model

In this paper, a test bed scheduling model bas been presented in which the resource selection criteria varies with the submitted job parameters, availability of worker nodes as well as computational parameter values of participating worker nodes at any scheduling point; thus the name adaptive scheduling. Figure 1 presents the major functional components of the Adaptive Scheduling Model and the following subsections provide their brief descriptions. Table 2 enlists the symbols used therein.

5.1. Job Analyzer and Job Decomposition:

Algorihtm:1 JobDecomposition

Input : It analyzes requirements of submitted independent jobs and arrives at tentative decomposition strategy
Output :Submitted job /jobs decomposed into set of tasks

$Job_{size=}\{T1_{size}, T2_{size}, T3_{size}, T4_{size} .....\}$

$TotJob_{size} = Job_{size}$

$do:$

$for\ (GridNode\ node_i\ :\ selectedGridNodes())$

$T_{size}(node_i) = \left(\sum_{node=1}^{N} AvgCPU_{Load_{node}} \Big/ job_{size} \right) * AvgCPU_{Load_{nodei}}$

$\mathrm{Re}\,qHeap_{Mem}(T_{size}(node_i)) = T_{size}(node_i) * \alpha_{Heap_{MemSize}}$

$if\ \left(\mathrm{Re}\,qHeap_{Mem}(T_{size}) > AvaHeap_{Mem_{node_i}}\right) then$

$\delta_{Job_{size}} = (\mathrm{Re}\,qHeap_{Mem}(T_{size}) - AvaHeap_{Mem_{node_i}}) / \alpha_{Heap_{MemSize}}$

$T_{size}(node_i) = T_{size}(node_i) - \delta_{Job_{size}}$

$endif$

$TotJob_{size=} TotJob_{size} - T_{size}$

$i = i + 1$

$endfor$

$\}while(TotJob_{size} > 0);$

The job decomposition block is responsible for deciding how to split user-submitted independent (parallel) jobs more effectively. To do this, it collects detailed job information, like job type job size; SLA criteria details, like deadline and number of resources from the job submission portal and current grid resource information like available resources (available parameter values) from the middle ware and average communication overhead from past history. From this information, the job decomposer

provides key information like splitting percentage and parameter for splitting so that parallel jobs can be effectively decomposed. These blocks take necessary information from GridGain to accomplish their function

Algorithm 1 presents in a nutshell the essential functionalities provided by the job analyzer and decomposer block. The inner 'if' and 'do..while' block are responsible for combing the CPU and heap memory availability in order to avoid communication overheads. Jobdecomposition() is applicable for independent jobs to improve the efficiency of overall system by aptly splitting the jobs into tasking according the resources availability of computational power.

This algorithm tries to find out the maximum task size that can be scheduled with available heap memory and available cup load of the resources. As desktop grids heterogeneous in nature, in which available CPU load doesn't make any sense in computation. So, average CPU load is considered for find the maximum task size to allocate a node. "if statement" take care of heap memory overflow. Once task size is decided then the challenging is to find the required heap memory for the task size. $\alpha_{Heap_{MemSize}}$ is calculated using table lookup approach, the details of table lookup technique is discussed in Section 4.5.

5.2. Computational Parameter Selection:

Algorithm: 2 ParameterSelection( ): to choose that parameter, scheduling based on which expected to give best performance

1. $Begin:$
2. $for\ \left(parameter\ P_i \quad = P_1\ to\ P_5 \right)$
3. $AvgJob_{size} = Avg\left(\sum\ Job_{size}\left[\sigma_{jobtype='usrJob' \cap Para = P_i}\ Job_{size}(GridExe_{Tab}) \right] \right)$
4. $AvgR_{size} = Avg\left(\sum\ R_{size}\left[\sigma_{jobtype='usrJob' \cap Para = P_i}\ R_{size}(GridExe_{Tab}) \right] \right)$
5. $AvgExe_{time} = Avg\left(\sum\ Exe_{time}\left[\sigma_{jobtype='usrJob' \cap Para = P_i}\ Exe_{time}(GridExe_{Tab}) \right] \right)$
6. $TempExe_{time}\left[P_i \right] = \dfrac{\left(AvgR_{size} * AvgExe_{time} * Job_{size} \right)}{AvgJob_{size} * n}$
7. $endfor$
8. $\min val = TempExe_{time}[P_1]$
9. $for\ P_i = P_2 to\ P_5$
10. $if\ \left(\min val > Exe_{time}[P_i] \right) then$
11. $Set \min val = Exe_{time}[P_i]$
12. $Set\ parameter = P_i$
13. $end\ if$
14. $End\ for$
15. $return\ parameter$
16. End

database for efficient functioning. Since, this method has to be invoked at every scheduling point; hence, the extracted data are kept as a temporary table. Table 3 depicts the necessary fields extracted and this is created as a part of GridGain initialization process by running a specified set of tasks. Algorithm 3 captures the essential features of the parameter selection method

5.3. Heuristic Scheduling

As grid resources are high heterogeneous in nature and distributed in geographical area, the grid scheduler should use some resource allocation strategy in terms of user requirements and job sorting methods according to the characteristics of sub-jobs (tasks), such as task size, task type. There are some work considering these parameters of sub-jobs and resources, here we considered parameters like job type, job size, available grid resources, their computational power, heap memory and past execution history.

Algorithm: 3 Heuristic Algorithm

```
---------------------------------------------------------------------------------------------------
Input  : Batch of tasks : TList; available resources list :RList
Output  : The Assignment of tasks to the resources
---------------------------------------------------------------------------------------------------
for each job ∈ TList
      sort tasks according to their size;
end
para=ParameterSelection( ExeLog, Jobtype);
for each resource ∈ RList
      sort resource list according to their para: computational parameter;
end
for i=1; i <= M; i++ // M is the number of available grid resources
      map the i_th task from TList to the resource RList
end
if N > M
      for j = M; j<= N; j++ // N is the number of jobs
            map the j_th task from TList to the resource from RList with least finish time
end
```

6. Performance Tuning Model

The adaptive scheduling model proposes a methodology for distributed execution of jobs within the desktop grid. But owing to the unpredictability of grid environment, scheduled tasks are susceptible to performance fall downs. Thus, for long running tasks, it become a prime necessity to monitor the execution and from time-to-time tune scheduling parameters for achieving tuning mechanism and Figure 2 shows its major functional components. The following subsections provide brief accounts of each of them.

6.1. Execution Analyzer

As user submitted jobs are scheduled to worker nodes and get executed, owing to the uncertainties associated with grids, performance may fall due to various reasons. This

necessitates a scheme for monitoring grid performance and track anomalies, if any. The execution analyzer block shown in Figure 2 is responsible for creating a list of tasks to be scheduled. Algorithm 3 summarizes its functionality.

6.2. Re-scheduler:

Algorithm 4: ReScheduling:Algorithm()

```
public Map<? extends GridJob, GridNode> map(List<GridNode> subgrid, GridifyArgument arg ) throws GridException {Map<GridJobAdapter<String>, GridNode> jobs = new HashMap<GridJobAdapter<String>, GridNode>(subgrid.size());
int JobStatus[][]=new int[NJobs][2];
int jobId=0;
---------- mapping of resources to jobs using AdaptiveGA----------
On assign a job to a machine
JobStatus[jobId][0]=arg.getid();
JobStatus[jobId][1]=0;
}
public String reduce(List<GridJobResult> results)throws GridException{
On receive sub result of any job with jobId
If(JobStatus[id][0]==jobId)
JobStatus[jobId][1]=1;
If(waitingTime > EET_job+ Δt ) then
For(i=0;i<jobList.getLength();i++)
If(JobStatus[i][1]==0)
   Reschedule, map failed jobs to best node available preferable at local.
Endif
Endfor
endi
}
end
}
```

This block is invoked whenever the execution analyzer indentifies performance degradation beyond certain set thresholds. Once invoked, the re-scheduler analyzes the available local and remote nodes for possible alternative nodes to re-schedule tasks. Subsequently re-scheduling is done with the goal of minimizing completion time of the set of submitted jobs. Algorithm 3 presents the internal functional of the rescheduling mechanism associated with the CPTF.

7. Experiments and Results

The previous sections present an in-depth coverage of the internal design of the CPTF, outlining the design requirements. This section intends to describe the experimental scenarios, the experiments carried out to assess the performance of the CPTF and subsequently present the results.

7.1. Environment

In an effort to study the performance of CPTF, the concepts discussed in previous sections have been implemented on a set of personal computers which forms an integral part of a PC based test bed deployed using GridGain 5.2. The desktop grid system has been setup using four programming labs of National Institute of Science and Technology, Berhampur. Each lab has about 450 PCs in it with different configurations. As these laboratories were setup at different times spanning over a period of roughly ten years, so the PC configurations differ from each other. Each node of all four labs has GridGain 5.2 installed and running on it with JDK- 6u-10, Java Runtime Environment and Eclipse 3.2 on them. Different constituent machines have different operating systems, like Microsoft's Window XP, Professional (service Pack 2), Ubuntu Linux 10.0.

High Performance Computing (HPC) systems exist to provide means for parallel processing of various CPU or otherwise resource intensive tasks. A common approach is to place any number of dedicated multi-core servers in close proximity to form a computing cluster. Traditionally the HPC approach involves splitting tasks into smaller work units and distributing such work units within cluster for parallel execution. This approach generally results in linear scalability, as with addition of new processing power, new work units can be easily created and the overall task execution will become proportionally faster.

A good example scenario would be the use of Monte Carlo Simulations across a wide spectrum of industries. In the financial sector, for example, the Monte Carlo method is used to assess the value of companies, perform risk analysis, or calculate financial derivatives. The method relies on repeated random sampling by running simulations multiple times in order to calculate the same probabilities heuristically. Problems that fit the Monte Carlo method can be easily split into multiple sample ranges, with each sample distributed across the computational cluster for parallel execution.

For cases when some grid nodes are more powerful or have more resources than others you can run into scenarios where nodes are not fully utilized or over-utilized. Under-utilization and over-utilization are both equally bad for a grid. Ideally, all grid nodes in the grid should be equally utilized. GridGain provides several ways to achieve equal utilization across the grid including:

Weighted Load Balancing

If you know in advance that some nodes are, for example, 2 times more powerful than others, you can attach proportional weights to the nodes. For example, part of your grid nodes would get weight of 1 and the other part would get weight of 2. In this case job distribution will be proportional to node weights and nodes with heavier weights will proportionally get more jobs assigned to them.

Adaptive Load Balancing

For cases where nodes are not equal and you don't know exactly how different they are, GridGain will automatically adapt to differences in load and processing power and will send more jobs to more powerful nodes and less jobs to weaker nodes. GridGain achieves that by listening to various metrics on various nodes and constantly adapting its load balancing policy to the differences in load.

Early and Late Load Balancing

GridGain provides both early and late load balancing for HPC load distribution, effectively enabling full customization of the entire load balancing process. Early and late load balancing allows adapting the grid task execution to non-deterministic nature of execution on the grid.

Early load balancing is supported via mapping operation of the MapReduce process. The mapping – the process of mapping jobs to nodes in the resolved topology – happens right at the beginning of task execution and therefore it is considered to be an early load balancing stage. Once jobs are scheduled and have arrived on the remote nodes for execution they get queued up on the remote node. How long this job will stay in the queue and when it's going to get executed is controlled by the job collision resolution – that effectively defines the late load balancing stage.

One implementation of the load balancing orchestrations provided out-of-the-box is a job stealing algorithm. This detects imbalances at a late stage and sends jobs from busy nodes to the nodes that are considered free right before the actual execution.Grid and cloud environments are often heterogeneous and non-static, tasks can change their complexity profiles dynamically at runtime and external resources can effect execution of the task at any point. All these factors underscore the need for proactive load balancing during initial mapping operation as well as on destination nodes where jobs can be in waiting queues

7.2. Experiments Conducted and the CPTF/ PES

Since the primary goal of this paper is to establish the superiority of the proposed CPTF that combines adaptive scheduling with rescheduling options over simple adaptive scheduling, hence all experiments conducted were targeted to find out the execution time (completion time) once employing only adaptive scheduling and then repeating the same by employing CPTF. In order to have an extensive assessment of their respective performances, the experiments user repeated with:

- Varying grid sizes (*i.e.*, by varying the number of PCs at an instant in the desktop grid).
- Varying the size (implying computational requirements) of submitted job.
- Varying the computational load of PCs currently participating in desktop grid.
- Varying the job mix, *i.e.*, by varying the ratio of dependent and independent jobs.
- Varying the scheduling mode, *i.e.*, batch and online mode.

Since, the number of combinations for all these variations will be very complicated to manually handle, the experiments were automatically repeated for different combinations and the execution times were stored in database. For the sake of simplicity, this setup for automatic performance evaluation was designed separately so that once the users submit their jobs, the performance of grid for these at that instant are observed and recorded. This is being referred to as CPTF Performance Evaluation Setup (CPTF/PES) and has been referred hereafter in abbreviated form as the CPTF/PES. For every set of experiments the user had to enter SLA parameters and subsequently the designed CPTF/PES would repeat the same for all per-set variations. Besides, the CPTF/ PES is designed such that for a job submitted to the system, for the same grid scenario, the jobs are run multiple times and the execution times are noted,

subsequently these normalized. For example, in order to assess the effect of job size on execution time for varying grid sizes, the different execution times obtained for various local load and job mixes are normalized to obtain a single figure of execution time.

The CPTF/PES randomly selects required number of nodes for scheduling based on the user input from the pool of available PCs in the consistent laboratories. Four job types have been used for this purpose, two of them being dependent jobs and the other two independent jobs (one that can be split to tasks before scheduling). Job sizes have been categorized as small, medium and large based on execution times. The CPTF/PES is pre programmed to classify a user submitted job into any one of the three possibilities. It is to be noted that this paper presents the results only for computational jobs, thus all jobs are computational intensive. Scheduling mode bas been chosen by user. In this paper, only jobs scheduled in batch mode are considered. Local load of PCs indicate local applications running on a PC and a predefined set of applications had been identified that correspond to light, medium and heavy CPU loads. While the CPTF/PES automatically switches between these load scenarios, it ensnare appropriate local application(s) that run on the activated (grid constituent) nodes by terminating previous ones and working new depending on current scenario. As far as job-mix is concerned, this is repeated manually for three sets once with majority of dependent jobs, then with majority of independent jobs and finally with equal mix of both types. This has been depicted by α and β in the next section.

7.3. Analysis of Results:

The CPTF/PES collects results in terms of execution time in milliseconds for all the various scenarios; once employing the adaptive scheduling mechanism solely and subsequently by employing the CPTF/PES framework. The results of these experiments have been presented in Figure 4(a) through 4(e) each with different grid size. The following subsections present the effects of variation different parameters on performance.

7.3.1. Effect of Grid Size on Performance:

Adaptive scheduling and CPTF exhibits scalability with increasing grid sizes as can be seen in the graphs shown in Figure 4(a) through Figure 5(a). For small jobs, though, this trend is reversed. CPTF, on the other hand, is observed to exhibit much more uniform scalability as can be seen in figures. The superiority of the CPTF over its adaptive counterpart is distinctively perceptible for smaller jobs. This can be ascribed to the fact that CPTF takes into account communication overhead prior to task decomposition. Adaptive scheduling presented in [15] suffers from discontinuity in scalability at some particular grid size, due to which increase in grid size decreases performance. In the experiments conducted with the CPTF/PES setup, CPTF exhibited around 10 - 15 % performance benefit for small jobs and 15 - 20% performance benefit for large jobs when compared to adaptive scheduling, for insignificant changes in other scenarios.

7.3.2. Effect of Local Load on Performance:

As can be observed from the graphs presented in Figures 4(a) through 4(e) [both adaptive and CPTF, CPTF doesn’t exhibit significant advantage over adaptive scheduling for the scenario where no extra local load is provided. But as local load of grid nodes increase, CPTF tends to outperform its adaptive counterpart. Performance

benefits vary between 10 to 20%, and it can be observed that this performance benefit tends to increase with job size, *i.e.*, large jobs on CPTF show better performance benefit over adaptive scheme than on small jobs.

7.3.3. Effect of Job Mix:

In this paper, both independent jobs that are decomposed to several sub jobs are used along with jobs that are scheduled without decomposing. Considering heterogeneous nature of grid jobs it can be assumed that overall grid job submitted are not biased, *i.e.*, total job submissions are equipoised (This is considered the base case of α =0.5, β=0.5). Figures 5(a) through 5(e) exhibit the effect of job mix on grid performance by plotting the performance of two ideal variant cases, namely heavily biased dependent jobs and heavily biased independent jobs with the base case (of an equipoised job mix). α and β represent the ratio of independent to dependent jobs at a scheduling point. For heavily biased dependent jobs, it has been considered that α =0.7, β=0.3 and for heavily biased independent jobs α =0.3, β=0.7 are considered. It has to be noted that the vertical axis shows the deviation in performance of these job mix cases from the base case of α =0.5, β=0.5.

It can be observed that for heavily biased dependent jobs, the trend of deviation in execution time as compared to the equipoised case of job mix (α =0.5, β=0.5) is much more pronounced, both for adaptive as well as for CPTF. However, from an overall perspective; CPTF outperforms adaptive scheme by showing smaller deviation from equipoised job mixes and the deviation has been observed to be even upto 30% - 40% when compared to adaptive scheme. Besides, the deviations in all cases show consistently larger deviations for peak local load scenarios.

7.3.4. Performance Comparison of Adaptive Scheduling of the CPTF

Figure 4(a) through 4(e) summarizes the performance benefits that CPTF provides over adaptive scheduling under all accounted scenarios for varying grid sizes. In these figures, S1 and S3 denote no local load and peak local load scenario respectively where as the labels large and small in the figures denote various job sizes. α, β corresponds to job-mix variations. The data points in the plots represent the normalized execution times for different variations. It has to be understood that the increasing radii of the concentric circles shown in these figures represent the increasing execution time. The choice of these spiral graphs over commonplace line graphs can be ascribed to its capacity of reducing the number line graphs.

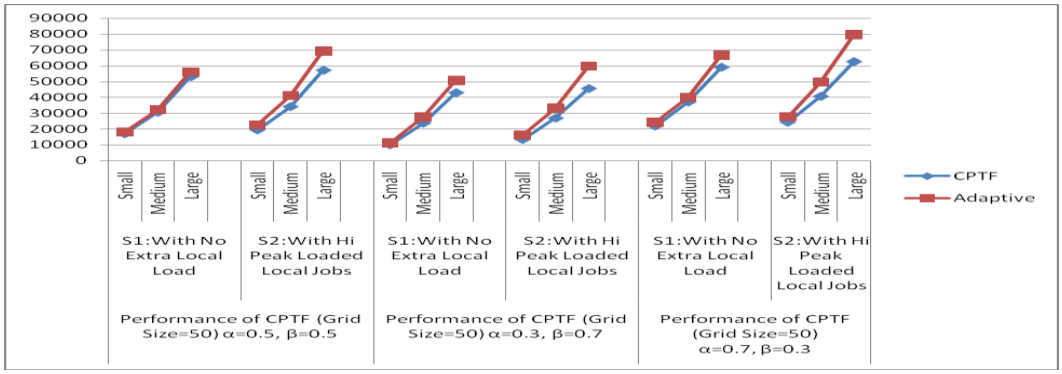

Figure 4. (a) Performance of Adaptive Scheduling Vs CPTF for Grid Size GridSize 50

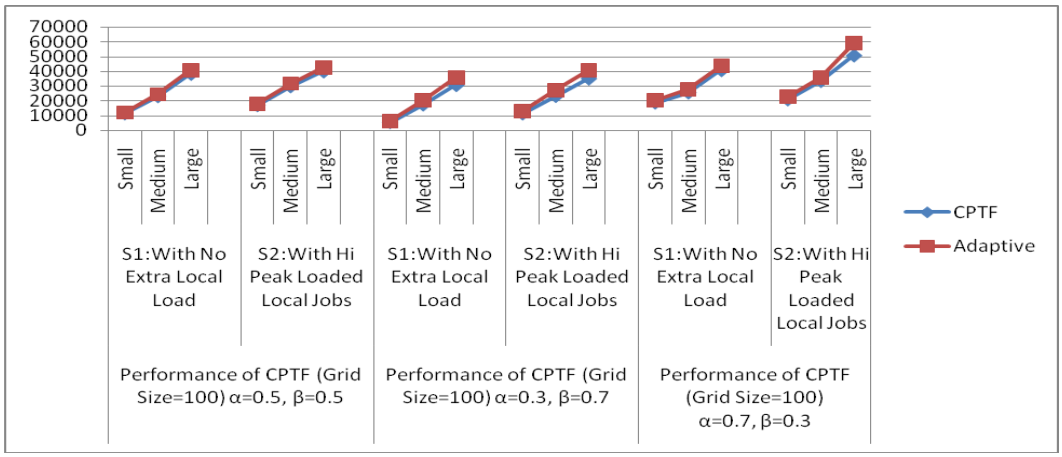

Figure 4. (b) Performance of Adaptive Scheduling Vs CPTF for Grid Size GridSize 100

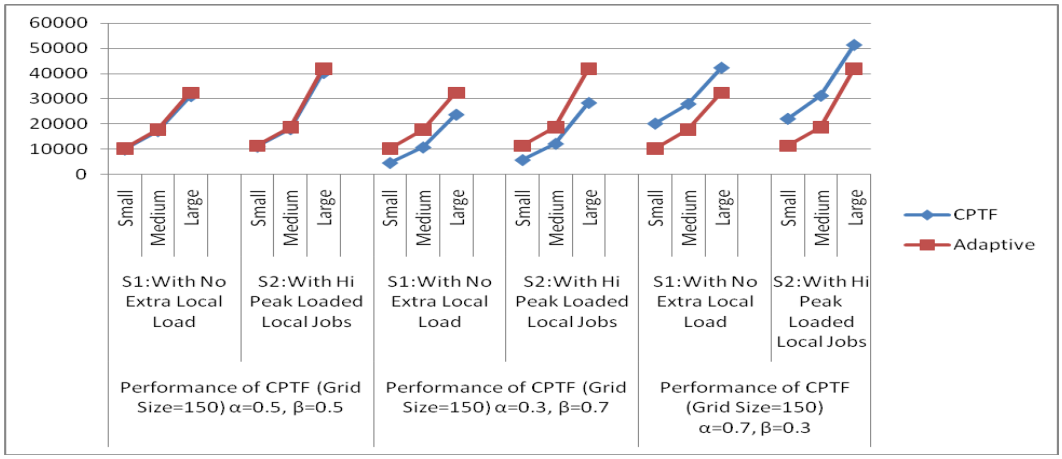

Figure 4. (c) Performance of Adaptive Scheduling Vs CPTF for Grid Size GridSize 150

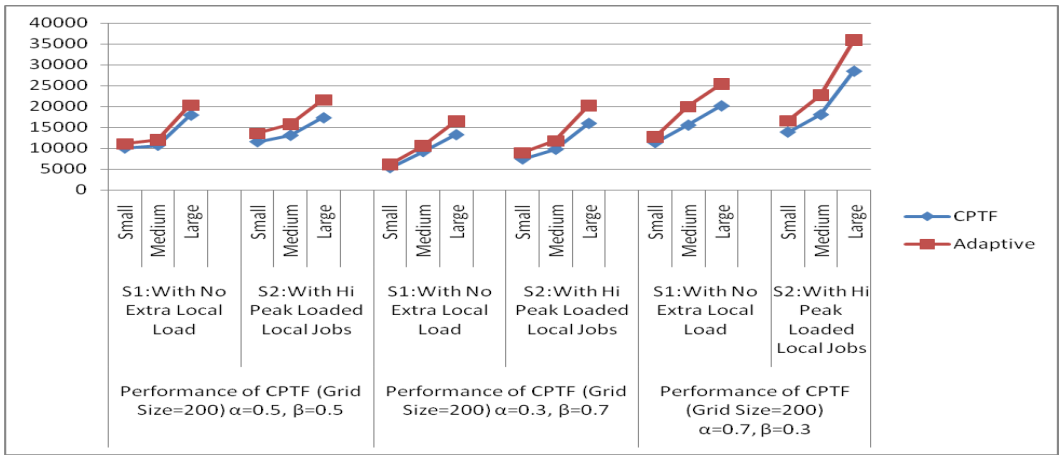

Figure 4. (d) Performance of Adaptive Scheduling Vs CPTF for Grid Size GridSize 200

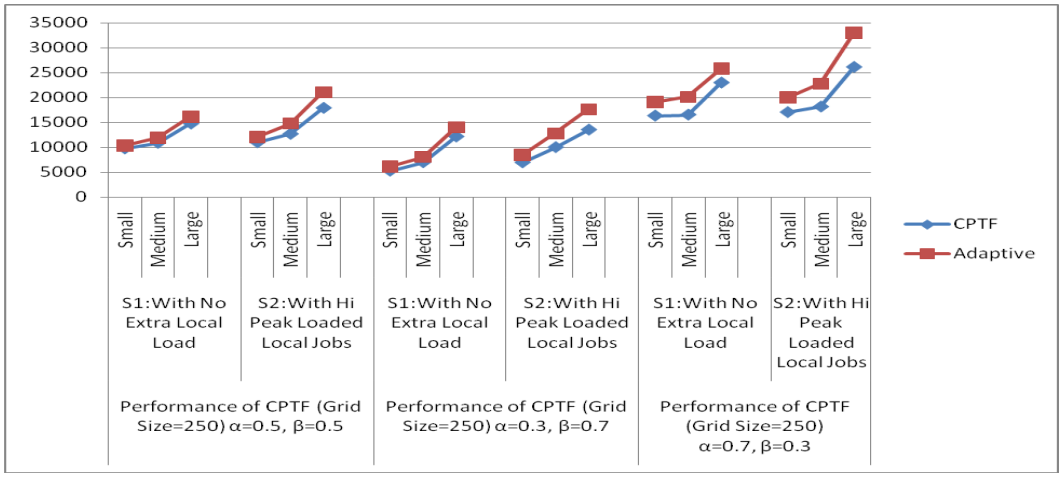

Figure 4. (e) Performance of Adaptive Scheduling Vs CPTF for Grid Size GridSize 250

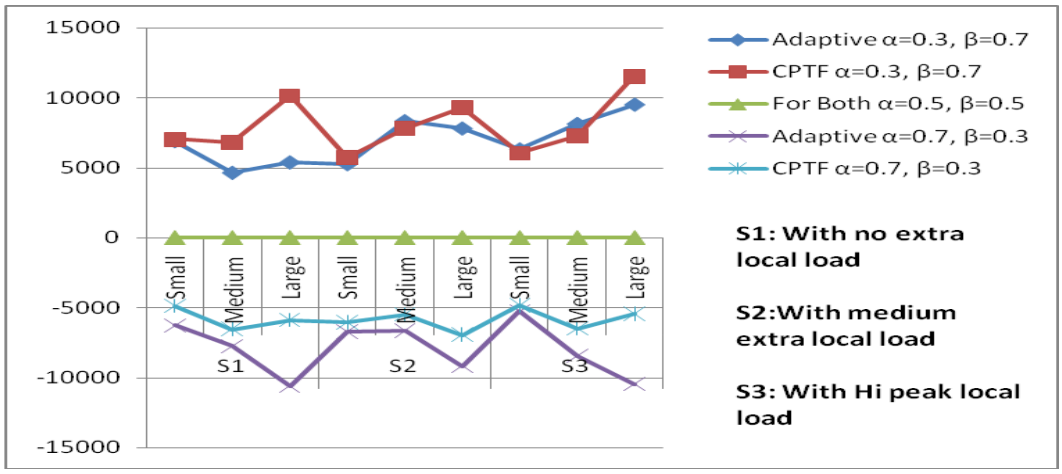

Figure 5. (a) Effect of Job-mix on Grid Performance: Grid Size 50

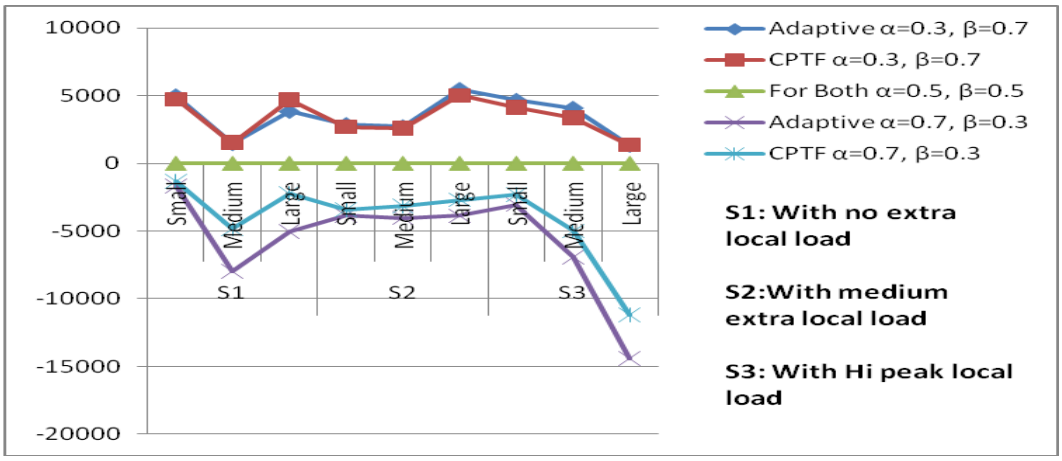

Figure 5. (b) Effect of Job-mix on Grid Performance: Grid Size 100

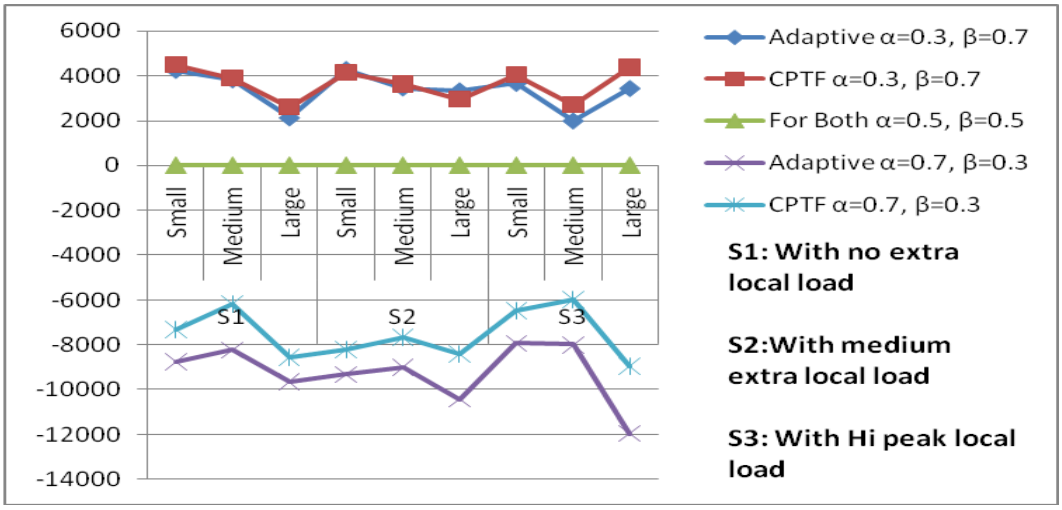

Figure 5. (c) Effect of Job-mix on Grid Performance: Grid Size 150

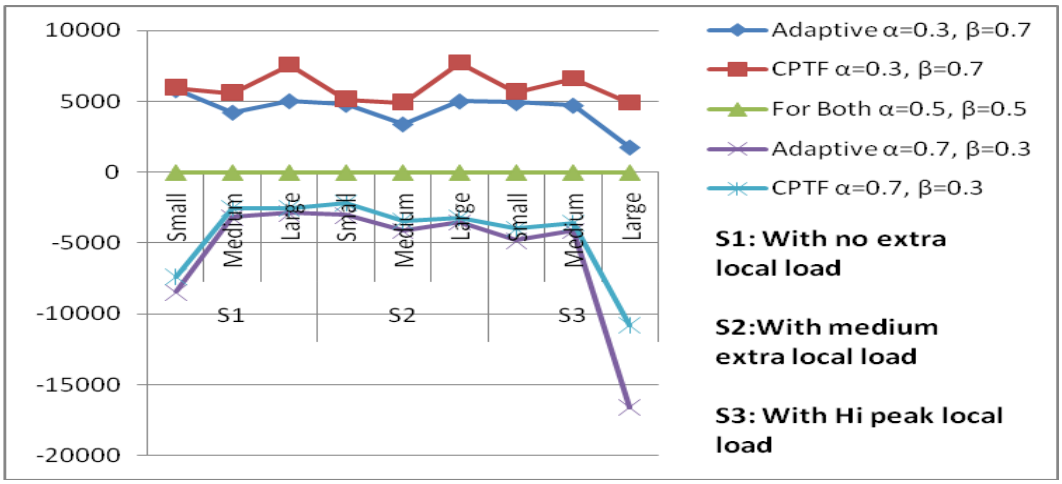

Figure 5. (d) Effect of Job-mix on Grid Performance: Grid Size 200

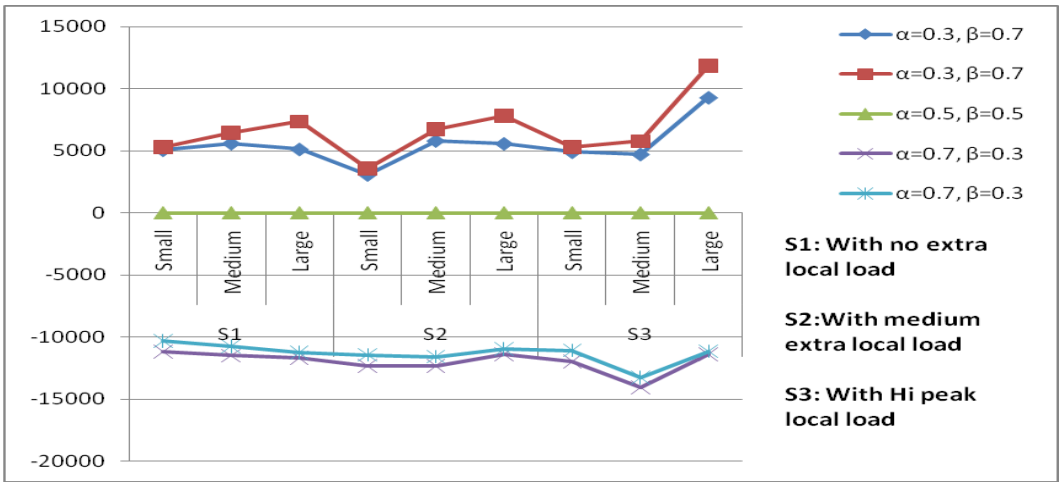

Figure 5. (e) Effect of Job-mix on Grid Performance: Grid Size 250

8. Conclusion

This paper presents a performance tuning framework for computational jobs in a desktop grid that employs an execution monitoring and performance tuning mechanism in addition to an adaptive scheduling scheme. The main emphasis of this paper is to provide an in-depth coverage of the requirements and design of the major elements of the CPTF with a detailed experimental study. While simulation study of grid scheduling approaches can seldom capture run-time intricacies or anomalies that may crop up during execution in a real test bed; test bed results can have limited exposure owing to the fact that only specific jobs are generally tested. In this paper, a real test bed deployment of the CPTF has been treated. Moreover numerous scenarios have been taken into account to capture the dynamic and unpredictable nature of desktop grids in order that a wholesome picture of CPTF's performance can be assessed including job sizes, available resources, local load of PCs and so on. In that sense, the CPTF/PES is more like an emulator than a real test bed. Results presented in this paper conclusively establish the superiority of CPTF over an adaptive scheduling model. This is very significant to particularly for desktop grid since the results clearly show that the overhead for performance tuning is surpassed by the benefits that come by using it. In this paper, all jobs have been considered to be computation intensive, which is a gross simplification of real life grid jobs. Thus issues related to effective data intensive jobs scheduling in the aforementioned framework becomes a natural research extension for future exploration

Acknowledgements

[1] I. Foster and C. Kesselman, "The Grid 2: Blueprint for a new computing infrastructure", Elsevier, **(2003)**.
[2] R. Buyya, C. Shin Yeo, S. Venugopal, J. Broberg and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility", Future Generation computer systems, vol. 25, no. 6, **(2009)**, pp. 599-616.
[3] C. Chen, J. Chame and M. Hall, "CHiLL: A framework for composing high-level loop transformations", U. of Southern California, Tech. Rep., **(2008)**, pp. 08-897.
[4] M. Schordan and D. Quinlan, "A source-to-source architecture for user-defined optimizations", Springer Berlin Heidelberg, **(2003)**.
[5] C. A. Schaefer, V. Pankratius and W. F. Tichy, "Engineering parallel applications with tunable architectures", Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ACM, vol. 1, **(2010)**, pp. 405-414.
[6] A. Hartono, B. Norris and P. Sadayappan, "Annotation-based empirical performance tuning using Orio", Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium, IEEE, **(2009)**, pp. 1-11.
[7] C. Ţapuş, I.-H. Chung and J. K. Hollingsworth, "Active harmony: Towards automated performance tuning", Proceedings of the 2002 ACM/IEEE conference on Supercomputing, IEEE Computer Society Press, **(2002)**, pp. 1-11.
[8] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta and W. Deng, "New grid scheduling and rescheduling methods in the GrADS project", International Journal of Parallel Programming, vol. 33, no. 2-3, **(2005)**, pp. 209-229.
[9] E. Huedo, R. S. Montero and I. M. Llorente, "Evaluating the reliability of computational grids from the end user's point of view", Journal of Systems Architecture, vol. 52, no. 12, **(2006)**, pp. 727-736.
[10] E. Huedo, R. S. Montero and I. M. Llorente, "A framework for adaptive execution in grids", Software: Practice and Experience, vol. 34, no. 7, **(2004)**, pp. 631-651.
[11] R. Sarbani, M. Sarkar and N. Mukherjee, "Optimizing resource allocation for multiple concurrent jobs in grid environment", Parallel and Distributed Systems, 2007 International Conference, IEEE, vol. 2, **(2007)**, pp. 1-8.
[12] R. Sarbani and N. Mukherjee, "Adaptive execution of jobs in computational grid environment", Journal of Computer Science and Technology, vol. 24, no. 5, **(2009)**, pp. 925-938.
[13] P.-C. Chen, J.-B. Chang, T.-Y. Liang, C.-K. Shieh and Y.-C. Zhuang, "A multi-layer resource reconfiguration framework for grid computing", Proceedings of the 4th international workshop on Middleware for grid computing, ACM, **(2006)**, pp. 13.

[14] M. Wu and X.-H. Sun, "A general self-adaptive task scheduling system for non-dedicated heterogeneous computing", Cluster Computing, 2003. Proceedings, 2003 IEEE International Conference, IEEE, **(2003)**, pp. 354-361.
[15] M. Chtepen, F. H. A. Claeys, B. Dhoedt, F. De Turck, P. A. Vanrolleghem and P. Demeester, "Performance evaluation and optimization of an adaptive scheduling approach for dependent grid jobs with unknown execution time", 18th World IMACS Congress (IMACS 2009) and MODSIM09 ICMS: Interfacing modelling and simulation with mathematical and computational sciences, pp. 1003-1009.
[16] http://www.gridgain.com/products/in-memory-hpc/.
[17] K. Reddy, K. Hemant, M. Ranjan Patra, D. Sinha Roy and B. Pradhan, "An adaptive scheduling mechanism for computational desktop grid using gridgain", Procedia Technology, vol. 4, **(2012)**, pp. 573-578.
[18] A. A. Azab and H. A. Kholidy, "An adaptive decentralized scheduling mechanism for peer-to-peer desktop grids", Computer Engineering & Systems, 2008. ICCES 2008. International Conference, IEEE, **(2008)**, pp. 364-371.
[19] S. Choi, H. Kim, E. Byun and C. Hwang, "A taxonomy of desktop grid systems focusing on scheduling", Department of Computer Science and Engeering, Korea University, Tech. Rep. KU-CSE-2006-1120-01, **(2006)**.
[20] F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems", Future generation computer systems, vol. 26, no. 4, **(2010)**, pp. 608-621.
[21] K. Hemant Kumar Reddy and D. Shina Roy, "A hierarchical load balancing algorithm for efficient job scheduling in a computational grid testbed", Recent Advances in Information Technology (RAIT), 2012 1st International Conference, IEEE, **(2012)**, pp. 363-368.

Authors

K. Hemant Kumar Reddy was born in India and received his MCA. M. Tech from Berhampur University in 2004 and 2008 respectively. He is currently with the National Institute of Science and Technology, Berhampur, India as an Assistant Professor. His research interests include distributed and grid computing, service oriented architectures, optimization in engineering.

Dr. Manas Ranjan Patra was born in India. He holds a Ph.D. degree in Computer Science from the Central University of Hyderabad. He has been teaching in the Post Graduate Department of Computer Science, Berhampur University since 1987. As a United Nations visiting Fellow he carried out research at the International Institute for Software Technology, United Nations University, Macau in the year 2000. He worked as an Assistant Professor at the Institute for Development and Research in Banking Technology (Reserve Bank of India), Hyderabad during 2005-06. Currently, he heads the Department of Computer Science, Berhampur University. He is a life member of CSI, ISTE & OITS, and a Fellow of ACEEE.

Diptendu Sinha Roy was born in India. He received his B. Tech from Kalyani University in 2003 and subsequently his M. Tech and Ph.D from Birla Institute of Technology, Mesra, India in 2005 and 2010 respectively. He is currently with the National Institute of Science and Technology, Berhampur, India as an Associate Professor. Dr. Sinha Roy's research interests include distributed and grid computing, software reliability, optimization in engineering. He also works towards design and analysis of distributed infrastructure of power systems.

