An Automated Performance Tuning Scheduling Framework for Computational Jobs in Desktop Grid

K Hemant Kumar Reddy
Department of Computer Science & Engineering
National Institute of Science and Technology, Berhampur, India

Diptendu Sinha Roy
Department of Computer Science & Engineering
National Institute of Science and Technology, Berhampur, India

Manas Ranjan Patra
Department of Computer Science
Berhampur University, Berhampur, India

Abstract- 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. Unfortunately few efforts have focused on tuning the performance of desktop grid scheduling at run time. Scheduling of jobs is a challenging proposition, and is even more true for a dynamic system like desktop computational grid. In comparison to scheduling decisions made strictly made based upon grid information, scheduling meta-heuristic mechanisms employing optimization techniques hold more promise for dynamically changing computational environments. We exploit this property and have proposed a modified Genetic Algorithm based scheduling and performance tuning framework. In this paper, the design and implementation of this Automatic Performance Tuning Scheduling Framework (*APTSF*) is reported that initially schedules jobs to resources using genetic algorithm, mines all such job-to-resource mapping information and thereafter tunes certain parameters for subsequent scheduling of submitted jobs. The unique contribution of the Heuristic based *APTSF* lies in formulating the same for a real-life test-bed, beyond the paradigm of simulation based models. Several experiments have been conducted on the test bed of desktop computers employing *GridGain* as middleware under varying load conditions, and comparative results of *ASF* and *APTSF* is presented to show the efficacy of the proposed framework.

Keywords – Grid scheduling, Heuristic scheduling, Performance Tuning,

I. 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 expertly 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. Due to high possible fluctuations in resources availability, network bandwidth and computational power (like CPU load, heap memory etc.) 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. The *APTSF* reported in this paper initially schedules jobs to resources, mines all such job-to-resource mapping information and thereafter tunes certain parameters for subsequent scheduling of submitted jobs. The philosophy of *APTSF* is based on the following capabilities (i) a parameter selection technique that selects the

best computational parameter, based on its value resources are collected (ii) an adaptive prediction approach based on which expected execution time is calculated (iv) a heuristic applied for scheduling the jobs to resources and (v) a post scheduling monitoring mechanism that identifies performance degradation beyond certain thresholds. In order to investigate the capabilities of the *APTSF*, a test bed of desktop computers has been deployed to set up a grid using *GridGain* 2.0. The work presented in this paper builds upon the Adaptive scheduling, performance tuning framework proposed and implemented in an earlier article [3], by embedding capabilities like rescheduling (migrating) jobs (or sub jobs) by means of appropriate tuning parameters. The detailed design of the *APTSF* has been presented in section 4. The results presented in this paper show the efficacy of the *APTSF* to tune performance for achieving better performance under varied network bandwidth as well as varied computation load among grid nodes.

The rest of this paper is organized as follows: Section II presents a brief outline of related work. Section III introduces the Automatic Performance Tuning Framework model along with a detailed discussion. Results and test-bed setup are presented and subsequently analyzed in section IV. Conclusions are presented in section V.

II RELATED WORK

There have been some researches covering performance tuning of applications on grid test beds. The Grads project integrated application monitoring and adaptive control services within their framework. They employed the Globus Toolkit [4] for middleware services to leverage effective scheduling policies for computation-intensive numerical solutions with rescheduling or. Huedo et.al. [5] 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 [6, 7] presented a framework (PRAGMA) where task migration and rescheduling of batch of jobs, both for super computer and clusters. A multi-layer resource reconfiguration framework for performance enhancement of user programs running on grid was proposed and implemented by Chen et.al. [8]. These frameworks needed adequate performance modeling. [9] presented self-adaptive grid scheduling without the need to use performance model. A.Y. Zomaya et.al. [10] have conducted and observed effect of heuristic algorithm for dynamic load balancing on parallel and distributed systems. Kalyanmoy Deb. Et.al. Similar to [11, 12], our work focuses on a resource scheduling strategy within a compute Grid for both dependent as well as independent jobs, but focus on adaptive scheduling algorithms proposed and implemented in an earlier article [3], by embedding capabilities like rescheduling tasks, improved job analyzer, decomposed technique and brokering for heterogeneous resources that are shared by multiple user jobs. The Heuristic Algorithm has extensively been used as a practical and robust optimization and search space method in various areas [13, 14, 15]. However our approach is different in that we design and apply a novel adaptive approach for the decomposition of a job into multiple sub-tasks while simultaneously considering communication overhead cost and computation cost.

III. THE AUTOMATED PERFORAMNCE TUNING FRAMEWORK

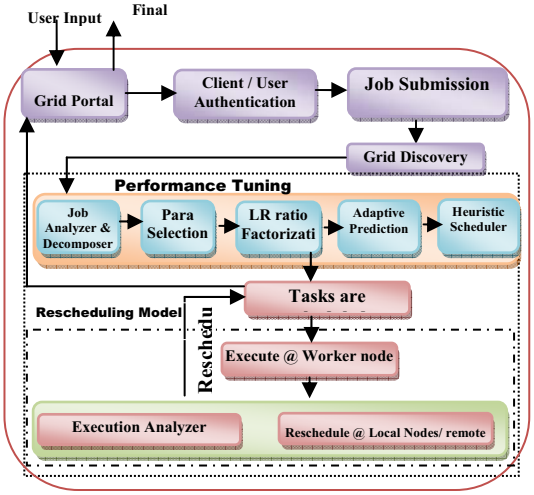

Figure 1: APTSF Model.

This section presents the *APTSF* framework that has been deployed at the *High Performance Computing (HPC)* lab at *National Institute of Science and Technology, Berhampur* campus [3], [17] along with other laboratories. Jobs can be submitted through any of the desktop PCs, which act as the grid portal. The heuristic scheduler is responsible for

mapping the tasks to appropriate resources within the participating grid nodes adaptively. This is done with the help of information provided by job analyzer and previous execution log information. The job scheduler model is thus adaptive, because the parameters based on which jobs are scheduled vary at run time. *APTSF* also provides the facility to constantly monitor execution status of all jobs and the execution history is mined. An execution analyzer is designed to identify glitches in performance and jobs can be rescheduled if needed. The following subsection provides a brief account of the steps that submitted jobs in the *APTSF* has to undergo. Figure 1 presents the major functional components of APYSF.

The aim of *APTSF* is to decrease job's completion time. In order to meet this goal, *APTSF* considers several parameters for effective scheduling of jobs. In most literatures [6, 8, 18], computing power of each resource in a grid is taken into account. *APTSF* takes into account computational power availability and heap memory availability during scheduling of jobs. It incorporates an adaptive scheme [19] for choosing appropriate parameters based on which resources are collected for mapping and adaptively predict the expected execution time, on which a modified genetic algorithm is applied for effective scheduling. As mentioned in section *III (D),* tuning an application can be manually accomplished knowing the target configuration and capacities, but such endeavors seem inappropriate for grid systems. Thus *APTSF* attempts to keep track of suitable job-resource mapping by mining performance data from previous runs. As in most service oriented computational schemes, *APTSF* 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.

The *APTSF* consists 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 resources are collected. Possible parameters are CPU availability, heap memory availability or the combined mean availability of these two parameters and so forth. A heuristic scheduler schedules the jobs on the basis effective execution times.

APTSF supports various modes of operation. It can allow job submission in an 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 sub-jobs depending on a criteria for reducing overall job(s) completion time 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 job related parameters and these parameters are different for batch mode and online mode of operation.

A. Job Analyzer and Decomposition

The job decomposition block is responsible for deciding how to split user-submitted independent (parallel) jobs effectively to decide task size for assigning them to a resource. To do this, it collects detailed job information, like job type, job size, number of resources available and expected time to complete from the job analyzer, current grid resource information like available resources (available parameter values) from the middle ware and average communication overhead from past history. It is quite logical to decompose independent jobs that are submitted online and similarly to keep the number of jobs submitted in batch mode as low as possible. Instead of executing a job completely in any one resource (while other resources are idle), it is better to decompose job(s) into tasks and distribute the tasks to available resources. But in this case the challenging task is to decide the splitting percentage and the number of tasks, which may lead to communication overhead. The job decomposition process has been discussed in the following section. These blocks takes necessary information from *GridGain* to accomplish their function, a briefing of which is provides as follows:

GridGain 2.0 middleware provides necessary information about the grid system, grid resource information and run time job execution details. The following grid run time features extracted using *GridGain* can be used to schedule the jobs. This necessary information from *GridGain* employed to accomplish their function has been briefly provided hereafter: Task size for a Resource (R_j) with respect to average available CPU load is given by

$$T_{Size_i} = \left(\sum_{J=1}^{N} AvlAvgCPU_{Load_J} \Big/ job_{size} \right) --(1) \quad T_{Size(R_j)} = \left(\sum_{J=1}^{N} AvlAvgCPU_{Load_J} \Big/ job_{size} \right) * \left(AvlAvgCPU_{Load_J} \right) --(2)$$

For parallel computational jobs to be decomposed only as per CPU availability, an unit of job can be formulated by (1) wherefrom the task size for a particular resource can be found out using (2). But user submitted independent (parallel) jobs can have markedly varying communication requirements for distributing its data for subsequent computation at the worker nodes. Thus, it is quite logical to incorporate the available heap memory status of resources while decomposing jobs for scheduling.

$$\mathrm{Re}\,qHeap_{Mem}(T_{size}(node_i)) = T_{size}(node_i) * \alpha_{Heap_{MemSize}} --(3)$$, where $\alpha_{Heap_{MemSize}}$ indicates the amount of heap memory required for task or job. This can be calculated with help of the number of data items it is using and their type involved in the job equation (4) calculates the required heap memory that is required for execution of a task or a job. Required heap memory size depends on job size and type of job.

$$\alpha_{Heap_{MemSize}} = \sum_{i=1}^{n} No_{DataType} * sizeof(\mathrm{int}) + No_{DataType} ---(4)$$

$$\mathrm{Req}Heap_{Mem_j} << AvlAvgHeap_{Mem}(R_j) --(5), \quad \mathrm{Re\,q}Heap_{Mem}(R_j) > AvalHeap_{Mem}(R_j) --(6)$$

Equation (3) estimates the amount of heap memory required for a task shown in (2) for a particular resource R_j . But combining the heap memory requirement of a resource with the estimated task size as per CPU availability, thresholds can be set for avoiding communication overhead conditions and memory overflow exceptions (as shown in (5)) as well as heap memory overflow(as shown in (6)).

B. Heuristic Scheduling Algorithm

In this paper an adaptive heuristic algorithm has been presented, in addition to dynamic information, keeps track of previous jobs' execution history to predict future jobs execution time in the aforementioned grid environment adaptively. The algorithm takes into account the processing capacity of the nodes, communication cost during the load balancing operation, heap memory requirement and pending jobs. The category of the problem we address here is: it is computation-intensive and the jobs are totally independent with no communication between them and dependent jobs. For the purpose of this work we have used the template given in Algorithm 1.

C. Problem formulation

In order to capture important characteristics of job scheduling in desktop grid systems, we considered the use of test-bed model. To formulate the problem under real test-bed model, an estimation of the computational load of each job, the computing capacity of each resource, job migration cost due to unreliable network bandwidth, and an estimation of the prior load of each one of the resources are considered to model estimation *Expected Execution Time (EET)model. EET* is matrix prepared adaptively with the help of previous execution log, which stores expected execution time of a job with respect to a resource.

In this paper, we consider the classical *minCompletionTime* Scheduling problem. We are given *'m'* machines for scheduling, indexed by the set M = {1, 2, 3, ..., *m*}. There are furthermore given *'n'* jobs, indexed by the set J = {1, 2, 3, ..., *n*}, where job j takes $Exe_{i,j}$ units of time if scheduled on machine *i*.

In this paper an attempt has been made to model a grid test-bed using heuristic algorithm and an adaptive prediction approach for predicting the *Expected Execution Time* (*EET*) for submitted jobs. The proposed model can be used to model the scheduling of job to resources with a multi-objective general formulation. The basic objective is that of minimizing the completion time that is, the time when the last job finishes its execution. Execution*Time* of Job_j in a Machine M_i is:

$$ExeTime(Job_j) = \left[Instructions(job_j) \Big/ Mips(M_i) \right] + Comm_{cost} \qquad (7)$$

Equation (7) works for grid simulation, but in case of test-bed desktop grid difficult to predict the execution time of a job. In this paper, an adaptive prediction approach is used to predict the expected execution time from execution log. To achieve this, training test benchmark computational jobs are executed in desktop and results stored in a log. This log information updated to a table during offline and table lookup approach is used to predict the expected execution time of online submitted jobs and expected execution time calculated in equation (8). Every time before starting the grid factory, $GridExe_{Tab}$ is updated by taking meaningful information from $GridExe_{Log}$. In a scalable grid system, the volume of the execution history can be prohibitively large enough to access any relevant data quickly. This can be a performance bottleneck for real time scheduler. To avoid this, a fixed set of meaningful log information is copied to $GridExe_{Tab}$. However, we have not considered the$GridExe_{Tab}$ update cost in our scheduling. Maximum completion time of Job_j in a machine Mi is calculated in equation (8)

$$MinExeTime = \min_{s_i \in Schdule} \left[\max_{j \in JobList} \left(ExpExeTime_j \right) \right] --\quad (8)$$

Where $ExpExeTime$ is the execution time of job$_j$, schedule is the set of all possible schedules, *jobList* the set of all jobs submitted for scheduling. In order to express the $ExpExeTime_j$ in *minExecutionTime* and formulation, an adaptive prediction approach used to calculate $ExpExeTime_j$ using table lookup approach. Execution time of job$_j$ can be predicted as follows:

$$AvgExe_{time} = Avg\left(\sum\left[\sigma_{jobtype='usrJob'}, Exe_{time}(GridExe_{Tab})\right]\right) --(11) \quad Comm_{cost} = \cos tTab\left(R_{type}, J_{type}, J_{size}\right); --(12)$$

$$AvgExe_{time} = \left[\frac{\left(AvgR_{PVal} * AvgExe_{time} * Job_{size}\right)}{AvgJob_{size} * n}\right] --(13), \quad ExpExe_{time}\left(Job_j\right) = \left\{\left(AvgExeTime * Job_{size_j}\right) \Big/ CurrR_{CP}\right\} --(14)$$

The costTab keeps the communication cost with respect to resource location (local or remote machine), jobsize, type of job and network bandwidth. This is calculated by considering a sample training data set on local nodes and remote nodes separately. This information is maintained in a table as shown in table 1. Communication cost and execution time are calculated using job execution time as per equations 9 and 10.

Table 1 costTab: communication cost Table

Resource site	Job Type	Job size	Network Bandwidth

$$CommCost = CNode_{STime} - DNode_{SendTime} --(14), \quad ExeTime = CNode_{STime} - CNode_{ETime} --(15)$$

This formula can be incorporated in different ways to establish the precedence among them. In the multi-objective heuristics approach two functional approaches are customized to fit the real test-bed model. Both approaches are considered in this work.

$$f(x): MinCompletionTime = c1 * MinCompletionTime + c2 * commCost; \quad --(16)$$

Where has a priori fixed after a preliminary tuning process as discussed in section.

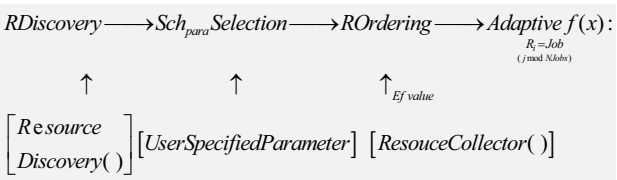

Figure 2(a) Control Flow of the Methods in *ASF*

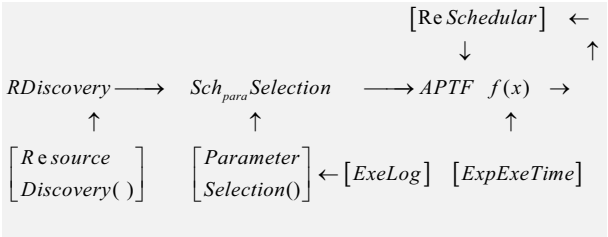

Figure 2(b) Control Flow of the Methods in *APTSF*

D. Rescheduling Approach

In dynamic scheduling scenarios, the responsibility for making automatic scheduling decisions is handled by centralized scheduler. In a computational Grid, there might be many applications submitted or required to be rescheduled simultaneously. The centralized strategy has the advantage of ease of implementation, but suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck [15, 16]. Sabin et al [17] propose a centralized meta-scheduler which uses backfill to schedule parallel jobs in multiple heterogeneous sites. Similarly, Arora et al [18] present a completely decentralized, dynamic and sender-initiated scheduling and load balancing algorithm for the Grid environment. To avoid this performance bottleneck a rescheduling mechanism proposed, which is a centralized reschedule.

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 1 is responsible for creating a list of tasks to be scheduled. 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 without terminating the prior one and considers the valid result of subtask which ever received first out of multiple map. The preference is given to idle local nodes first for rescheduling to minimize the communication cost. *GridFactory* aggregate the sub tasks results to form the complete result. This is to minimize the completion time of the set of submitted jobs.

Algorithm 1: Heuristic Algorithm()

```
1.  Begin
2.  Integer NodeCount=0; parameter=ParameterSelection( );
3.  for GridNode   node : AllGridNodes
4.        GridList.add(node);NodeCount++;
5.  endfor
6.   GridList=Selection(Gridlist,parameter);
7.  //----Job decomposer on criteria-----//
8.  if(NodeCount  > Njobs)
9.        taskList=jobDecomposer(Job);
10.       taskLen=taskList.length( );
11. else
12.       taskLen=Njobs;
13. endif    //---Map specific task to specific node -----//
14. Set Jcount =0, JobIndex=1;
15. for (k=1;k<=taskLen; k++)
16. minval=0;Gn=0;Job=0;
17. for(i=1;i<=NodeCount; i++)
18.       for (j=1;j<=taskLen; j++)
19.          if(min>EET[i][j])
20.         {minval=EET[i[[j];Gn=i;Job=j;} endfor
21. endfor
22.       Job.execute(GridList[Gn],taskList[Job]);
23.       set Jobindex=Jobindex+1;
24.  endfor
25.  set   jobExeIndex[count][o]=Jobindex;
26.            set   jobExeIndex[count][1]=0;
27.           set Jcount=Jcount+1;
28. Endfor
29. RSTParameterTuning( );
//--On completion of assigned task using Reduce function--//
30.  jobExeIndex[Jobindex][1]=1;
31. End.
```

Algorithm 2:ParameterSelection()

// -loop repeat for four parameters that identified -//

$$for\ parameter,\ P_i = P_1\ to\ P_3$$

$$TotJob_{size} = \sum Job_{size}\left[\sigma_{jobtype='usrJob' \cap Para=P_i} Job_{size}(GridExe_{Tab})\right]$$

$$TotR_{size} = \sum R_{size}\left[\sigma_{jobtype='usrJob' \cap Para=P_i} R_{size}(GridExe_{Tab})\right]$$

$$TotExe_{time} = \sum Exe_{time}\left[\sigma_{jobtype='usrJob' \cap Para=P_i} Exe_{time}(GridExe_{Tab})\right]$$

$$count++;\ endfor$$

$$AvgJob_{size} = TotJob_{size} / count$$

$$AvgR_{size} = TotR_{size} / count \ \&\ AvgExe_{time} = TotExe_{time} / count$$

$$TempExe_{time}[P_i] = \frac{(AvgR_{size} * AvgExe_{time} * Job_{size})}{AvgJob_{size} * n}$$

$$\min val = TempExe_{time}[P_1]$$

for P_i=P_2 to P_3

$$if\ (\min val > Exe_{time}[P_i])\ then$$

$$Set \min val = Exe_{time}[P_i]$$

$$Set\ parameter = P_i$$

end if

End for

return parameter

End.

IV GRID TEST-BED SETUP

In an effort to study the performance of desktop grid, in this work we have implemented an heuristic based scheduling algorithm on the desktop grid using *GridGain* 2.0. The desktop grid system has been setup with eight programming labs of National Institute of Science and Technology, Berhampur. Each lab has around 60 PCs in it with same configuration, but different configuration then other labs. As these labs were setup at different times spanning over a period of roughly ten years. Each node of all the four labs has *GridGain* 2.0.0 installed and running with JDK- 6u-10 and 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, and Ubuntu Linux 10.0.

A. Analysis of Results

The APTSF/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 APTSF/PES framework. The results of these experiments have been presented in figure 3 (a) through 3(d) each with different grid sizes. Every figure has three parts, the upper part showing the results of adaptive and APTSF scheme with α =0.3 & β =0.7 value whereas the middle part shows the normalized results (α=0.5 & β=0.5) and lower part shows the results with α =0.7 & β =0.3 value. The upper and lower part shows the difference from the middle part (α=0.5 & β=0.5). Upper and lower part shows the performance difference between *ASF* and *APTSF*. The following subsections present the effects of variation different parameters on performance.

B. Effect of Grid Size

ASF exhibits scalability with increasing grid sizes as can be seen in the graphs shown in figure 3 (a) through 3(d). For small jobs, though, this trend is reversed. APTSF, on the other hand, is observed to exhibit much more uniform scalability as can be seen in figures. The superiority of the *APTSF* over its *ASF* counterpart is distinctively perceptible for smaller jobs. This can be ascribed to the fact that *APTSF* takes into account communication overhead prior to task decomposition. *ASF* scheduling 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 *APTSF/PES* setup, *APTSF* exhibited around 10 - 15 % performance benefit for small jobs and 15 - 20% performance benefit for large jobs when compared to *ASF* scheduling, for insignificant changes in other scenarios.

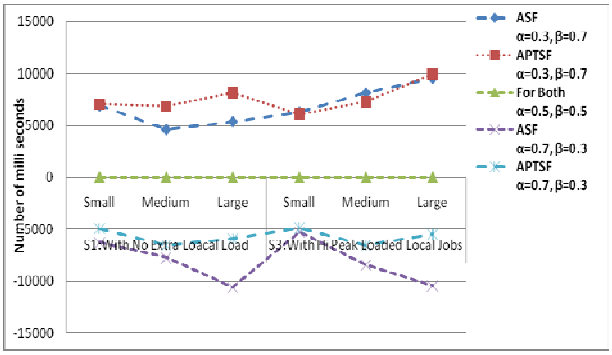

Figure 3 (a) Grid Performance for Grid Size 50

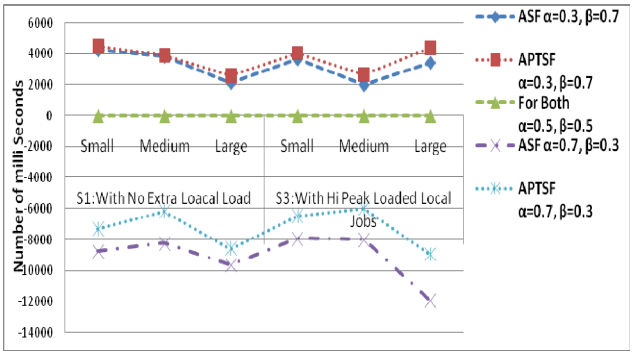

Figure 3(b) Grid Performance for Grid Size 100

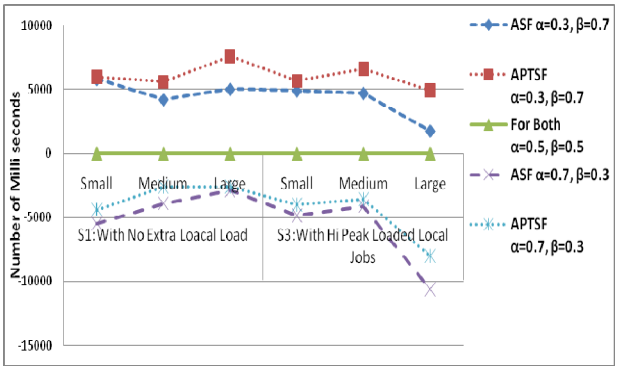

Figure 3 (c) Grid Performance for Grid Size 150

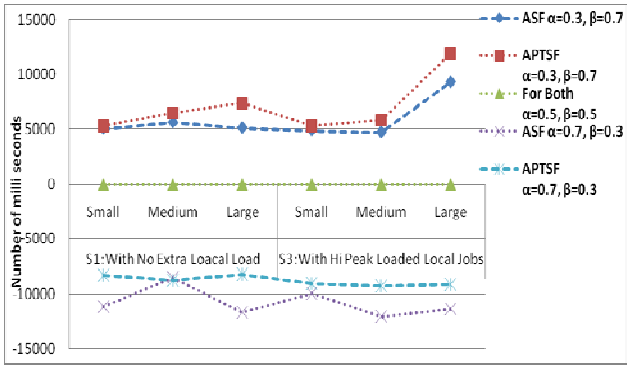

Figure 3(d) Grid Performance for Grid Size 200

This section presents the efficacy of different *ASF* and *APTSF*. Figure 3(a) through figure 3(d) shows the efficacy of *ASF* and *APTSF*; in which x-axis indicates execution time in milli seconds and y-axis present grid size.

v.CONCLUSION

The main emphasis of this paper is to study the effectiveness of Heuristic Algorithms for modeling an efficient desktop scheduling framework. To frame a performance tuning desktop grid scheduling model an *ASF* is tailored to fit in the desktop grid. Performance can be achieved by minimizing the *minExecutionTime* and *minCompletionTime*. In this paper, a real test bed deployment of the *APTSF* and *ASF* 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 *APTSF*'s performance can be assessed including job sizes, available resources, local load of PCs and so on. In that sense, the *APTSF*/PES is more like an emulator than a real test-bed. Results presented in this chapter conclusively establish the superiority of *APTSF* over an *ASF* scheduling model. This is very significant 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.

REFERENCES

[1] Foster, C. Kesselman. GRID2: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers, An imprint of Elsvier, 2006.

[2] Foster, C. Kesselman. Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufman, 1999.

[3] K. Hemant K. Reddy, Manas Patra, Diptendu Sinha Roy, B. Pradhan, "An Adaptive Scheduling Mechanism for Computational Desktop Grid Using *GridGain*", Procedia Technology, Elsevier, Volume 4, pp. 395 – 400, 2012

[4] S. Venugopal, R. Buyya, and L. Winton. A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids. In 2nd International Workshop on Middleware for Grid Computing, Middleware 2004, Toronto, Ontario - Canada, ACM Press, New York, NY, USA, October 18, 2004.

[5] Eduardo Huedo, Rube´n S. Montero, Ignacio M. Llorente, "Evaluating the reliability of computational grids from the end user's point of view", Journal of Systems Architecture 52 (2006) 727–736.

[6] Roy, S., & Mukherjee, N. (2009). Adaptive execution of jobs in computational grid environment. *Journal of Computer Science and Technology*, *24*(5), 925-938.

[7] De Sarkar, A., Roy, S., Ghosh, D., Mukhopadhyay, R., & Mukherjee, N. (2010). An adaptive execution scheme for achieving guaranteed performance in computational Grids. *Journal of Grid Computing*, *8*(1), 109-131.

[8] Chen, P. C., Chang, J. B., Liang, T. Y., Shieh, C. K., & Zhuang, Y. C. (2006, November). A multi-layer resource reconfiguration framework for grid computing. In *Proceedings of the 4th international workshop on Middleware for grid computing* (p. 13). ACM.

[9] M. Chtepen, F.H.A. Claeys, B. Dhoedt, F. De Turck, P.A.Vanrolleghem, P. Demeester. Performance evaluation and optimization of an adaptive scheduling approach for dependent grid jobs with unknown execution time. 18th World IMACS / MODSIM Congress, pp. 1003-1009. Cairns, Australia 13-17 July 2009.

[10] A.Y. Zomaya and Y.H. Teh. Observations on using heuristic algorithms for dynamic load-balancing. IEEE Transactions On Parallel and Distributed Systems, 12(9):899–911, 2001.

[11] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan," A Fast and Elitist Multiobjective Heuristic Algorithm: NSGA-II", IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 6, NO. 2, APRIL 2002.

[12] S. Song, Y. Kwok, and K. Hwang, Security-Driven Heuristics and A Fast Heuristic Algorithm for Trusted Grid Job Scheduling, in Proc. of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), pp.65-74, Denver, Colorado USA, April 2005.

[13] Yu, J., & Buyya, R. (2005). A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, *3*(3-4), 171-200.

[14] Dong, F., & Akl, S. G. (2006). Scheduling algorithms for grid computing: State of the art and open problems. *School of Computing, Queen's University, Kingston, Ontario*.

[15] Sabin, G., Kettimuthu, R., Rajan, A., & Sadayappan, P. (2003, January). Scheduling of parallel jobs in a heterogeneous multi-site environment. In *Job Scheduling Strategies for Parallel Processing* (pp. 87-104). Springer Berlin Heidelberg.

[16] Arora, M., Das, S. K., & Biswas, R. (2002). A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments. In *Parallel Processing Workshops, 2002. Proceedings. International Conference on* (pp. 499-505). IEEE.

[17] Reddy, H. K., Patra, M., & Roy, D. S. (2012, December). Adaptive execution and performance tuning of parallel jobs in computational desktop grid using GridGain. In *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on* (pp. 199-204). IEEE.

[18] Roy, S., Sarkar, M., & Mukherjee, N. (2007, December). Optimizing resource allocation for multiple concurrent jobs in grid environment. In *Parallel and Distributed Systems, 2007 International Conference on* (Vol. 2, pp. 1-8). IEEE.

[19] Choi, S. J., Baik, M., Gil, J., Park, C., Jung, S., & Hwang, C. (2006, May). Group-based dynamic computational replication mechanism in peer-to-peer grid computing. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on* (Vol. 2, pp. 8-pp). IEEE.