SLA Constrained Adaptive Scheduling of Parallel Jobs in a Computational Desktop Grid

K Hemant Kumar Reddy
National Institute of Science & Technology
Berhampur

Manas Ranjam Patra
Berhampur University
Bhanja Vihar
Berhampur

Diptendu Sinha Roy
National Institute of Science & Technology
Berhampur

ABSTRACT
Desktop grids are usually equipped with hundreds or thousands of desktops which use the idle cycles of desktop PCs of small enterprises and institutions. From conventional multi -site cluster grids, desktop grids vary markedly in terms of their dynamic nature. In terms of their dynamic nature such grids vary markedly from conventional grids] .This causes the need of new scheduling algorithms ,tailor-made for such systems .Such desktop grids become worthwhile research issue since the nature of these grids attract highly parallel algorithms ,designing efficient scheduling algorithms of parallel jobs on desktop grids .A SLA based adaptive scheduling framework (ASF) for desktop grids has been presented in this paper ,where based on a resource selection algorithm after SLA verification that set by the users tasks are assigned to different available grid resources .Based on a given set of parameter and previous execution log in an adaptive(flexible) manner the resource selection algorithm ,which works in an online mode, selects resources .Whether the user submitted jobs can be executed within the deadline and other constraints specified in the SLA or not the execution log decides .Choosing the most relevant computational parameter depending upon the tasks that are assigned to grid resources is also decided by it during task execution. Later, if performance of any task degrades during the task execution, then adaptive algorithm assists the scheduling policy to adapt the system to achieve the high throughput by re-scheduling to either the server node or to the best available local node at present. Unpredictable execution conditions commonly encountered on desktop grids which can violate the SLA constraints set by the user can also be handled by this method. In this paper, for setting up desktop grid test bed GridGain 2.0 has been used and the performance of the aforesaid SLA based adaptive scheduling framework has been presented.

Keywords
Adaptive execution; desktop grid; GridGain; rescheduling; performance tuning; service level agreement.

1. INTRODUCTION
For large-scale sharing and cooperation grid Computing deals with dynamically distributed resources, such as CPU-cycles, communication bandwidth and data [1]. Many scientific applications need tremendous computational power, ideally that requires employing expensive supercomputing facilities. For computationally intensive scientific application Dstp grid computing provides a low-cost alternative. On the other hand, as a computational desktop grid exhibits dynamic and unpredictable behavior; namely, the computational performances of each node vary significantly from time to time; the network connections may become unreliable; nodes may join or leave the grid system at any time; nodes may become unavailable without any notification this gives rise to significant challenges. As a result a wide range of completion times can be possessed by a computational job running on different nodes on the grid. Moreover, owing to the dynamic nature of desktop grids, the same job might also have different completion times on the same set of resources at different runs. Therefore, to minimize job execution time proper scheduling of the grid resources is an issue of prime importance. So there is a need of distributed computing which can provide huge computational infrastructure as well as highly available resources by the ever growing demand for computational requirements of these applications.

With the area of our work there are some relevant projects that have close links includes: In most recent years, when providing network services [2, 3, 4] or as means to make agreements between service provider and service consumer [5] SLAs became more pervasive. The trend in Grid system, application service delivery to move away from tightly coupled systems towards structures of loosely coupled, dynamically bound systems [6]. led to the implementation of SLA as a standard concept by which work on the Grid can be allocated to resources and enable coordinated resource management. The current understanding of the community in the context of Grid and Web services is that such an SLA is essentially an important contract, which is expected to be negotiated fully automatically by different processes. As a result, on various topics related to SLAs there has been continuous effort made from last decade. In [7, 8, 9] issues related to their overall incorporation into grid architectures have been discussed. In [10, 11, 12] issues related to the specification of the SLAs have been considered. In [13, 14, 15] issues related to the usage of SLAs for resource management have been considered. For self-adaptive mechanism on the grid that does not require performance model for resource selection reference [16] presented a methodology. Adaptive scheduling mechanism implemented in our earlier work and test-bet results analyzed [17].A performance evaluation and optimization model of an adaptive scheduling approach for dependent grid jobs, where the main focus is on computational and network overhead of the adaptive approach under different circumstances is proposed by M. Chtepen et. al. [18] .

In this paper for desktop grid environment we present an SLA constrained adaptive scheduling framework, which incorporates both the features of adaptive and SLA. For distribution of tasks to the best available resources an adaptive scheduling approach implemented .SLA guarantees that within the deadline the accepted job can be executed .In GridGain 2.0 middleware this framework has been implemented and experiment results have been discussed subsequently. The rest of the paper is organized as follows: Section 2 discusses desktop grid computing, Section 3

discusses the Desktop grid architecture, Section 4 presents the proposed algorithm and discusses the implementation details of the proposed algorithm, and the grid test-bed setup presented in section 5 and results of the paper are presented in section 6

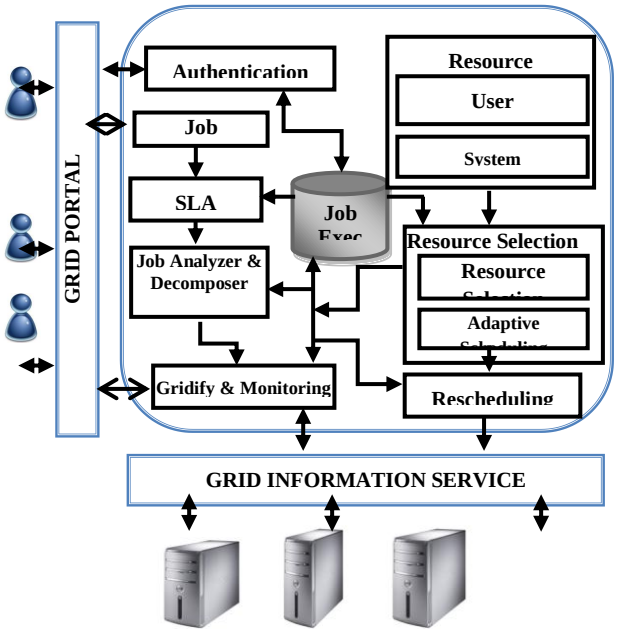
2. Desktop Grid Computing

For high throughput applications Desktop Grid has recently become a smart computing paradigm. However, as it is based on desktop computers from the view of internet [3, 4] desktop grid computing becomes convoluted by heterogeneous capabilities, failures, volatility, and lack of trust. A desktop grid computing environment mainly consists of clients, worker nodes, and a server. The job of a client is to acts as a parallel job submission portal. A worker node acts as a resource provider that donates its computing resources. The job of a server is to act as the central manager for controlling the submitted jobs and resources. To a server a client submits a parallel job. A job is divided into sub-jobs, called tasks. Using a scheduling mechanism the server allocates tasks to worker nodes. While continuously requesting data from its server each worker node executes its task. It returns the result of the task to the server when each worker node subsequently finishes its task. Finally, it returns the final result of the job back to the client when the server collects all results of tasks from worker nodes.

As compared to cluster grids scheduling in desktop grids differs, since a desktop grid may vary widely depending upon the type of resources, dedication, trust, failure modes, applications, and so forth. The process of assigning jobs to the most appropriate resources is known as Grid scheduling. In two ways scheduling may be performed *i.e.*, in a centralized fashion or in a fully distributed way [5]. For the most part, unlike clustered grids desktop grid systems do not need any local scheduler since in this case the scheduling target is a single desktop computer, contrary to a site in Grid [6, 7]. Heterogeneous, volatile, faulty, and malicious resources make Desktop grid scheduling complicated. Volatility (non-dedication), lack of trust, and heterogeneous properties are the main focus areas of Desktop grid scheduler than grid scheduler [8]. As compared to cluster computers or super-computers in desktop grid, communication cost is very high. In hierarchical desktop grid environment, as compared to cluster computers or high performance system, it is very difficult to classify grid resources into groups. 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. The autonomy of worker nodes (that is, worker nodes can freely participate in public execution) is respected by Desktop grid. Much like in peer to peer systems a test-bed model that provides distributed scheduling we have used in this work . A grid test bed has been deployed using GridGain for the purpose of demonstrating the efficacy of the proposed SLA constrained adaptive scheduling model.

3. Desktop Grid Architecture

Desktop grid framework that was developed at the High performance Lab, at NIST campus is presented in this section. The architecture of this grid frame work shown in figure1.



3.1 Desktop Grid Framework

For scheduling the submitted user jobs, the scheduler just go through three phases, namely 1Job submission-SLA verification, 2 Resource Discovery, 3 Resource Selection, 4 Job scheduling and Monitoring.

I. Job Submission –SLA Verification

a. Job Submission: The first step of scheduling process and job submission phase, where along with job specification like job type, job size and user submits user specification.

b. SLA Verification: where user submits SLA constraints specification like deadline, mode of execution and number of resources for execution.

II. Resource Discovery Phase:

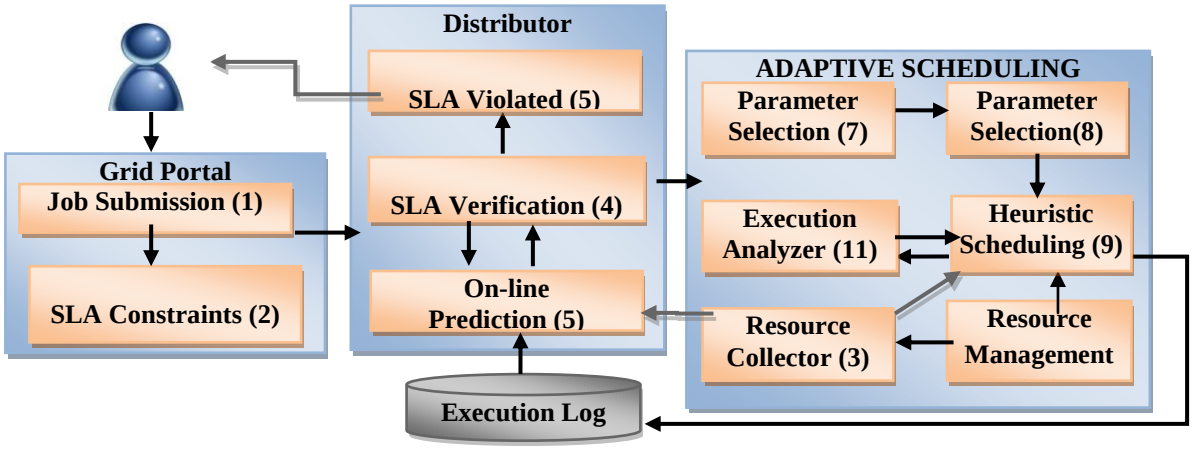
a. Application Specification**:** The first step of resource discovery phase, where for a specific job users need to specify some minimum requirement criteria, it includes static and dynamic information regarding job assigned and grid resources (e.g; Job, Job size & others)

b. Resource Collection and selection**:** The next step is to discover the list of available resources along with resource information; further in resource selection this information will be for mapping of task to specific grid resources.

III. Job Scheduling and Monitoring phase:

a. Job Distribution: In this phase, the first step is a specific resource should be assigned by a task. The Grid factory assigns the tasks to specific resources based on resource selection information for execution.

b. Job Execution Monitoring: In this phase, the second step is to monitor the execution process and reschedule the task that fails to complete within a stipulated time period due to resource failure or resource overload with other tasks. Scheduler algorithm reschedules the failed task to an available resource in both the cases that best fits to the task at that moment



3.2 Desktop Grid Scheduling Model

Any user submitted job in the above mentioned desktop grid framework goes through several states before it can be scheduled to a worker node and is executed subsequently. Figure 2 depicts the different states through which a job is scheduled on the desktop grid. The users submit their jobs to the grid system through a portal, which eventually forwards all such submitted jobs to 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 a log file for scheduling of future jobs, the detailed description and the life cycle for adaptive scheduling presented in figure 1 [17, 18, 19]. In this paper SLA constrained implemented based on which distributor decides using a online prediction technique for execution of submitted jobs. In this paper we propose an effective job decomposition method considering parameters like number of grid resources available and their computational capacities and a parameter selection algorithm selects parameter based on which tasks distributed. To decide upon that parameter selection which is expected to give better performance we have used the previous execution history stored in logs for distribution. Corroboration of this fact occurs by the title SLA constrained adaptive scheduling in desktop grids. The grid distributor takes the responsibility of mapping the tasks to the various resource respective results computed to the distributor once the adaptive function decides upon a schedule, consisting of task-resource matches,. As per the map-reduce paradigm this consists of a reduce step. The grid distributed aggregates all such sets results and by means of the portal send them to the user.

4. SLA CONSTRAINED ADAPTIVE SCHEDULING

A. *Adaptive Scheduling in Desktop Grid*

In this paper, we present a test bed scheduling framework for which the resource selection criteria varies with the submitted job parameters, availability of worker nodes as well as computational parameters values of participating worker nodes at scheduling points. Thus, the title adaptive scheduling framework. The working of the proposed adaptive scheduling algorithm has been presented in the following sub section.

B. *The Scheduling Mechanism*

i. Job submission with SLA Constraints.
ii. Job Decomposition.
iii. Parameter selection
iv. Adaptive scheduling algorithm

i. *SLA Constraints.*

In this paper, we use expected completion time of a user submitted application as a SLA constraint. If the expected completion time is less than the deadline submitted by the user, only then a job is allowed for execution. This is shown by (5). Equations (1) through (4) define the parameters used by (5). Otherwise *the GridDistributor* rejects user submitted job.

$$AvgExe_{Time}(Job_j) = \frac{\left(\sum \left[\sigma_{jobtype='usrJob'} Exe_{Time}(GridExe_{Tab})\right]\right)}{\sum(\sigma_{jobtype='usrJob'} Count(GridExe_{Tab}))} --(1)$$

$$Comm_{cost} = costTab\left(R_{type}, Job_{type}\right); \quad --(2)$$

Communication cost is the cost that incurred to travel the required data and code segment from server node to worker or client node. This can be extracted during job execution time. Equation 10 and 11 formulate the execution time and communication cost from run time grid environment.

$$ExpExe_{time} = \left(AvgExe_{Time}(Job_j) + \Delta t\right) --(3)$$

$$ECT = \left(PJobExe_{Time}(R_i) + Comm_{cost}(R_{type}, Job_{type}) + ExpExe_{Time}\right) --- \ (4)$$

$$if\ (ExpComp_{time} < UsrDeadline_{time}) \quad (5)$$

ii. *Job Decomposition Logic*

Job decomposition is the first and foremost step of scheduling mechanism in our computational grid framework. According

to resource size and the heterogeneity of grid environment, user submitted jobs are decomposed into a number of tasks and these tasks are mapped to available resources for execution. The process of job decomposition and distribution is depends on the type of job and the parameter based on which scheduling decision will be taken. In our work, we implemented a computational desktop grid framework with GridGain 2.0 as middleware. In scheduling, user submitted job(s) are decomposed into number of tasks. For decomposition a simple job splitter technique employed, which considers some other parameter like job size, job type, number of resources and its parameter values like available heap memory and available CPU load. There exists no universal logic that is suitable for all type of grid environments [19, 20]; similarly there is no single universal job decomposition logic suitable for all type of jobs.

Available CPU load of a Resource R*j*, (6) is provided by GridGain

$$AvaCPU_{Load_j} = 1 - CPU_{Load_j} \quad (6)$$

The Currently used Heap memory of Resource R*j* (7) is

$$Heap_{usedMem_j} = TotHeap_{Mem_j} - AvaHeap_{Mem_j} \quad (7)$$

Task for unit of CPU load can be calculated as follows (8):

$$job/unit(AvaCPU_{Load}) = \left[\sum_{node=1}^{N} AvaCPU_{Load_{node}} \Big/ job_{size} \right] \quad (8)$$

$$T_{Size(node)} = job/\sum_{i=1}^{n} R_i = \left(\sum_{node=1}^{N} CPU_{Load_{node}} \Big/ job_{size} \right) * \left(AvaCPU_{Loadj} \right) \quad (9)$$

Task size for a resource (R_j) solely as per CPU availability is as follows (9):

$$JobExe_{Time} = JobExe_{STime} - JobExe_{ETime} \quad --(10)$$

$$Comm_{Cost} = (Server_{SendTime} - Server_{RecvTime}) - JobExe_{Time} \quad --(11)$$

iii. Parameter Selection Logic

Once resource information are collected for scheduling, a resource selection algorithm can be used for selecting the resources based on some computational parameters, like CPU load, Heap memory, number of pending jobs *etc.*. In this paper an execution based adaptive resource selection technique has been employed for selecting a parameter, based on which resources are selected for efficient scheduling (12) represents the following information extracted from the execution log. For performance reasons, this information is collected as a part of the GridGain initialization process.

$$\sigma\ Jobsize, Rsize,\ Exetime, SchP(GridExe_{DB})_{jobtype='job'} \quad (12)$$

Following are the steps for parameter selection:

i. *for parameter,* $P_i = P_1$ *to* P_3

ii.

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

iii. *end for*

iv. *minval=* $MinFun(Exe_{time}[P_i])$

v. *return parameter[minval];*

iv. Adaptive Scheduling Algorithm

This subsection presents the adaptive scheduling algorithm. The essence of this scheduling has been represented within the *for* loop encompassing steps *vii* to *x*.

//-------Jobdecomposer method to split the submitted jobs---//

i. taskList=jobDecomposer(Job);
ii. taskLen=taskList.length( );
iii. parameter=parameterSelection( );
iv. NewGridList=DoOrdering(GridList taskLen, paramter);
//----Assigning specific task to specific node ----//
v. Set count =1,index=0;
vi. for (GridNode node: NewGridList)
vii. Job.execute(node, taskList[index])
viii. set index=index+1;
ix. set jobExeIndex[count][0]=index;
x. set jobExeIndex[count][1]=0;
xi. endfor
//--On completion of assigned task---//
xii. jobExeIndex[count][1]=1;
xiii. End.

5. Grid TESTBED SETUP

In an effort to study the performance of desktop grid, in this work we have implemented an adaptive scheduling algorithm on the desktop grid. The grid scheduling algorithms have been deployed on a set of personal computers which is an integral part of a PC based test bed deployed using GridGain 2.0. The desktop grid system has been setup using four programming labs, each having about 50 to 60 PCs with different configuration. As these labs 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 2.0.0 installed and running on it with JDK-6u-10 and Java Runtime Environment and Eclipse 3.2 on them.

6. RESULTS

This section presents the results obtained by conducting experiments on the desktop grid. We have employed matrix multiplication as user applications. The job sizes vary from 1000 to 5000, where a job size of *n* denotes a submitted job of *n* x *n* matrix. Figures 3 and 4 show the performance of our adaptive scheduling mechanism as the execution time (in milli-seconds) for various jobs having different job sizes under varying load conditions. Figure 5 shows the same under high local load condition and compares our adaptive scheduling strategy with a non-adaptive alternative. The execution time of a job is calculated in equation (5).

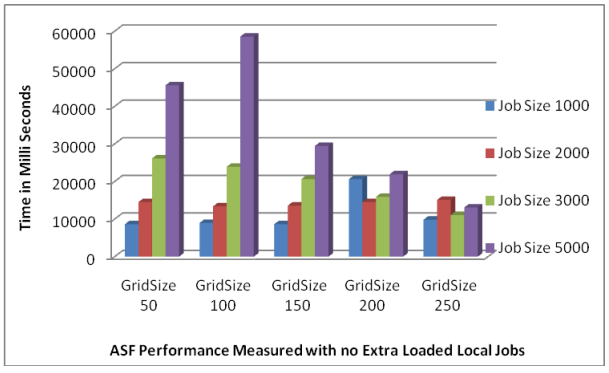


Figure 3 ASF Performance of varying grid and job size

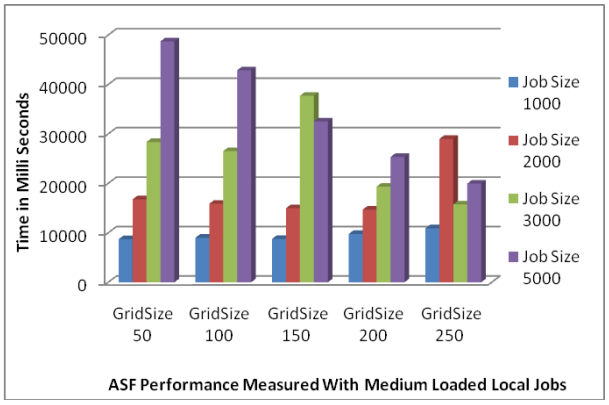


Figure 4 ASF Performance of varying grid and job size

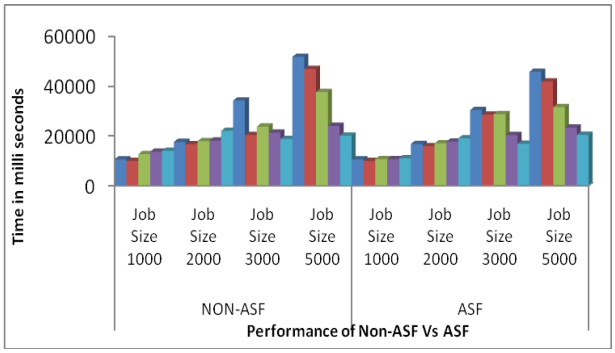


Figure5 Adaptive Vs Non-adaptive Scheduling Performance of varying grid and job size with high local load scenario

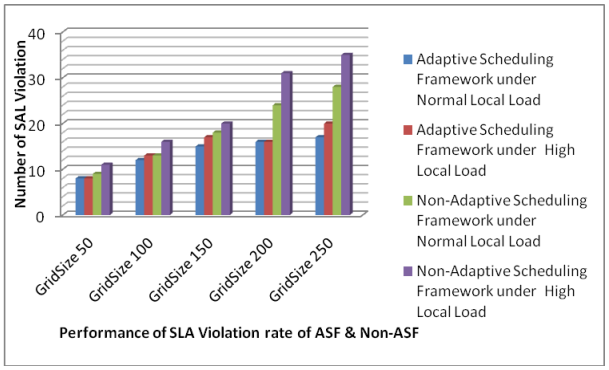


Figure 6 SLA Violation Rate for Varying Job size and Condition

Figure 6 shows the SLA violation rate with respect to job sizes. Separate sets of results are shown for various resource capacities under varying local load scenarios.

7. CONCLUSIONS

This paper provides a formulation of SLA constrained adaptive scheduling mechanism for desktop grids. In our approach, we employ a SLA validation mechanism to decide upon an acceptance scheme by comparing calculated execution time with user provided deadline. Results from test bed experiments have revealed the efficacy of our adaptive scheduling algorithm as presented in figure 5. Also the results presented in figure 6 shows that SLA violation rate increases for a highly loaded grid for large size jobs. Same is the trend for small sized jobs when large amount of grid resources are available. Currently we are in the process of fine tuning the results presents herein by using machine learning techniques on the execution log data that is expected to reduce the scheduling overhead considerably.

8. ACKNOWLEDGMENT

This work has been carried out at the HPC Lab, Department of Computer Science and Engineering, *National Institute of Science and Technology*, Berhampur.

9. REFERENCES

[1] Foster, Ian, and Carl Kesselman, eds. *The Grid 2: Blueprint for a new computing infrastructure*. Morgan Kaufmann, 2003..

[2] D'Arienzo, M., et al. "The service level agreement manager: control and management of phone channel bandwidth over Premium IP networks."*Proceedings of the 15th international conference on Computer communication*. International Council for Computer Communication, 2002.

[3] Verma, Dinesh. *Supporting service level agreements on IP networks*. New Riders Pub, 1999.

[4] Kagklis, Dimitrios, Nicolas Liampotis, and Christos Tsakiris. "Architecture for the creation of service level agreements and activation of IP added value services." *Telecommunications, 2003. ConTEL 2003. Proceedings of the 7th International Conference on*. Vol. 2. IEEE, 2003.

[5] Lee, Matthew KO. "IT outsourcing contracts: practical issues for management."*Industrial Management & Data Systems* 96.1 (1996): 15-20.

[6] Keller, Alexander, et al. "Managing dynamic services: A contract based approach to a conceptual architecture." *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*. IEEE, 2002.

[7] Mobach, David GA, Benno J. Overeinder, and Frances MT Brazier. "A resource negotiation infrastructure for self-managing applications." *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*. IEEE, 2005.

[8] Naik, Vijay K., Swaminathan Sivasubramanian, and Sriram Krishnan. "Adaptive resource sharing in a web services environment." *Proceedings of the 5th ACM/IFIP/USENIX international Conference on Middleware*. Springer-Verlag New York, Inc., 2004.

[9] Padgett, James, Mohammed Haji, and Karim Djemame. "SLA management in a service oriented architecture." *Computational Science and Its Applications–ICCSA 2005*. Springer Berlin Heidelberg, 2005. 1282-1291.

[10] Ludwig, Heiko, et al. "A service level agreement language for dynamic electronic services." *Electronic Commerce Research* 3.1-2 (2003): 43-59..

[11] Ludwig, Heiko, et al. "Web service level agreement (WSLA) language specification." *IBM Corporation* (2003): 815-824.

[12] Burchard, L-O., et al. "The virtual resource manager: an architecture for SLA-aware resource management." *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*. IEEE, 2004.

[13] Dumitrescu, Catalin L., and Ian Foster. "GRUBER: A Grid resource usage SLA broker." *Euro-Par 2005 Parallel Processing*. Springer Berlin Heidelberg, 2005. 465-474..

[14] Yarmolenko, Viktor, et al. "SLA based job scheduling: A case study on policies for negotiation with resources." *AHM*. 2005.

[15] Wu, Ming, and Xian-He Sun. "A general self-adaptive task scheduling system for non-dedicated heterogeneous computing." *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*. IEEE, 2003.

[16] Reddy, K. Hemant K., et al. "An Adaptive Scheduling Mechanism for Computational Desktop Grid Using GridGain." *Procedia Technology* 4 (2012): 573-578.

[17] Chtepen, Maria, et al. "Performance evaluation and optimization of an adaptive scheduling approach for dependent grid jobs with unknown execution time." (2009): 1003-1009.

[18] Wu, Linlin, Saurabh Kumar Garg, and Rajkumar Buyya. "SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments." *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. IEEE, 2011.

[19] Yarmolenko, Viktor, and Rizos Sakellariou. "An evaluation of heuristics for sla based parallel job scheduling." *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006.

[20] Yarmolenko, Viktor, et al. "SLA based job scheduling: A case study on policies for negotiation with resources." *AHM*. 2005.