International Journal of Web-Based Learning and Teaching Technologies
Volume 15 • Issue 1 • January-March 2020

High Performance Fault Tolerant Resource Scheduling in Computational Grid Environment

Sukalyan Goswami, Institute of Engineering & Management, Kolkata, India
Kuntal Mukherjee, Birla Institute of Technology, Mesra, Lalpur Campus, Ranchi, India

ABSTRACT

Virtual resources team up to create a computational grid, which is used in computation-intensive problem solving. A majority of these problems require high performance resources to compute and generate results, making grid computation another type of high performance computing. The optimization in computational grids relates to resource utilization which in turn is achieved by the proper distribution of loads among participating resources. This research takes up an adaptive resource ranking approach, and improves the effectiveness of NDFS algorithm by scheduling jobs in those ranked resources, thereby increasing the number of job deadlines met and service quality agreements met. Moreover, resource failure is taken care of by introducing a partial backup approach. The benchmark codes of Fast Fourier Transform and Matrix Multiplication are executed in a real test bed of a computational grid, set up by Globus Toolkit 5.2 for the justification of propositions made in this article.

KEYWORDS

Computational Grid, Fault Tolerance, Resource Failure, Resource Management, Task Scheduling

1. INTRODUCTION

Task scheduling in appropriate resources and achieving balanced load in computational grid environment (Foster, Kesselman & Tuccke, 2001) are the two major research areas which need to be explored more. This is mainly because of heterogeneous nature of grid and technological requirement of computation-data separation. Computational capability of grid can be enormous if participating resources remain well coordinated by broker and effective task scheduling is highest prioritized work for achieving balanced load across grid.

Nearest Deadline First Scheduled (NDFS) algorithm (Goswami & Das, nee De Sarkar, 2014) solves the above mentioned problems after the efficiency of the algorithm has been improved by considering average load of each resource along with its current load for a pre-defined interval. Saaty's Analytic Hierarchy Process (AHP) (Saaty, 2008) has been adapted and improvised to rank the participating resources. Subsequent job scheduling and load balancing in grid are completed by NDFS, thus meeting the Service Quality Agreement (SQA) (Goswami & Das, nee De Sarkar, 2014). Another objective of this research work is to make NDFS more robust by taking care of sudden possible occurrences of resource failure in grid. Periodical runtime backup to next adaptively ranked resource ensures compliance of approved SQA, which was signed between broker and client. Even if resource fails, job need not be resubmitted in fault tolerant NDFS, submitted job resumes execution

DOI: 10.4018/IJWLTT.2020010104

from last backed up point, into next ranked resource. Hence, job is submitted by client only once, thereby reducing total execution time. This paper puts thrust on single and multiple job execution in real grid environment through execution of benchmark codes, namely, Fast Fourier Transform and Matrix Multiplication. The experimental results are obtained for demonstration of balanced workload in computational grid along with optimal solution for resource failure by creating test bed of computational grid through Globus Toolkit, and comparing performance with other algorithms in GridSim (Buyya & Murshed, 2002).

The organization of this paper is as follows: after discussing relevant works by different researchers on grid in section 2, improvised NDFS algorithm is presented in section 3. Section 4 depicts results of execution of benchmark codes in grid and discussion on the obtained results. The paper is concluded in section 5.

2. RELATED WORKS

There are many challenges associated with computational grid, like, scheduling of jobs in appropriate resources for subsequent execution, increasing the number of deadline meets for submitted jobs, balancing of workload in grid and fault tolerance. Important related works in this arena of research are summarized in this section.

The load balancing process must take into consideration the dynamic loads of participating resources of the grid (Abo Rizka & Rekaby, 2012). Job scheduling has to ensure higher number of deadlines meets, thereby improving the performance of the grid. In (Goswami & Das, 2016), an adaptive execution scheme has been proposed which ensures guaranteed performance with respect to service quality agreements.

The properties of general distributed system is quite different from that of computational grid environment. Moreover, the client-server framework of grid proposed by Michael Stal (Stal, 1995) does not solve the problem of load balancing among participating resources in grid. Various job scheduling schemes in grid have been described in (Kant Soni, Sharma & Kumar Mishra, 2010). X-Dimension binary tree data model (Abo Rizka & Rekaby, 2012) have limited success in workload balancing among grid resources.

This research work documents an approach to improve the performance of computational grid by reducing the the number of job deadline misses and overall makespan in grid. Job scheduling in appropriate resources, being a multi-criteria-decision-making (MCDM) process, AHP model (Saaty, 2008) of Saaty, supporting MCDM, is augmented in this research. The broker schedules submitted jobs in adaptively ranked resources, ranking being done with help of AHP model. This adaptive scheduling ensures performance improvement of grid by subsequently reducing number of deadline misses and workload balancing among resources. Deadline of individual job is chosen as the prioritization parameter for job scheduling in Nearest Deadline First Scheduled (NDFS) (Goswami & Das, 2015) algorithm, and GridSim (Buyya & Murshed, 2002) simulation results show that NDFS improves grid performance. Another milestone achieved is the number of submission by single client, one job needs to be submitted once only by client, need of re-submission by client is done away with, broker takes care of the possible re-submission in different resource in grid and monitors previously signed SQA between client and broker. Resource ranking of NDFS has been refined in this research by incorporating average load of the resource.

Resource failure creates possibility of deadline misses in grid. Multiple fault tolerant approaches in grid have been explored in (Goswami & Das, nee De Sarkar, 2014). Author's earlier research in (Goswami & Das, nee De Sarkar, 2014) dealt with resource failure by introducing resubmission approach which had incurred overhead. This research work improves the performance of NDFS by incorporating partial backup approach without diluting performance of the grid.

3. NDFS ALGORITHM WITH FAULT TOLERANCE

3.1. Proposed Model of Resource Management

NDFS algorithm performs job allocation in grid by adaptively ranking the participating resources. Computational grid environment hits a critical scenario when number of jobs submitted by clients in grid out-numbers the number of resources, and the proposed model in this work of enhanced NDFS with fault tolerance deals with this critical situation successfully.

Following discussion describes the working of NDFS.

3.2. Job Submission and Resource Addition in Grid:

Client submits 5 parameters of itself along with job – number of processor cores, current CPU utilization, CPU clock frequency, current utilization of network and current RAM availability.

$$\text{\# of Jobs submitted to grid: } J_x,\ 0 \leq x \leq n \tag{1}$$

$$\text{\# of Resources available in grid: } R_y,\ 0 \leq y \leq m \tag{2}$$

n and m are maximum no. of executable jobs and maximum no. of resources that could be supported by grid, respectively.

$$\text{Assumptions: } J_x > R_y \tag{3}$$

3.3. Processing Phases

Job execution by NDFS is initiated after SQA is signed between client & broker, which happens after Job Queue is prepared based on $Job_{weightage}$ and adaptive resource ranking is complete based on $Resource_{weightage}$. Saaty's AHP model (Saaty, 2008) formulates calculation of the two valuable parameters $Job_{weightage}$ and $Resource_{weightage}$.

3.4. Allocation

$Job_{weightage}$ and $Resource_{weightage}$ need to be mapped for allocation of optimum number of jobs into optimum number of available resources in grid, eventually maximum number of SQAs are met.

$$\text{Assignment: } Max(J_x) \supset Min(R_y) \tag{4}$$

$$Job_{weightage}\ (J_a) <= Resource_{weightage}\ (R_b)$$

SQA verification,

$$J_x(C_t) <= J_x(dl) \tag{5}$$

where, $J_x(C_t)$ = completion time of job, J_x
$J_x(dl)$ = deadline of job, J_x

Successful allocation implies that, jobs (J_x) are scheduled to appropriate resources (R_y) and SQAs are met.

3.5. Compliance of SQA

The broker processes 3 phases after client successfully submits job in grid:

(i) Preparation of job queue
(ii) Ranking of resources
(iii) SQA endorsement

3.6. Preparation of Job Queue

$Job_{weightage}$ is calculated and resource ranking is obtained by the broker using AHP decision matrix (Goswami & Das, 2016).

From [14],

$$f(Job_{weightage}) = 0.464x_1 + 0.195x_2 + 0.195x_3 + 0.073x_4 + 0.073x_5 \quad (6)$$

3.7. Ranking of Resources

Resource ranking in computational grid has to support dynamicity, as because loads of participating resources continuously change. So, CPU Availabilty parameter gets amended to accommodate both current load and average load of the resource:

$$\text{CPU Availability}(x_2) = (0.5 * \text{current availability} + 0.5 * \text{average availability}) \quad (7)$$

Hence, the equation (6) is redefined as presented in equation (8):

$$f(Resource_{weightage}) = 0.464x_1 + 0.0975x_{21} + 0.0975x_{22} + 0.195x_3 + 0.073x_4 + 0.073x_5 \quad (8)$$

Broker calculates $Resource_{weightage}$. Highest $Resource_{weightage}$ valued resource is ranked as $Res_{rank} = 1$, next high $Resource_{weightage}$ valued resource gets $Res_{rank} = 2$ and so on. According to their ranks, resources are placed on a priority queue by broker.

3.8. SQA Endorsement

On finding higher value of $Resource_{weightage}$ compared to $Job_{weightage}$ of submitted job, bipartite agreement SQA is signed between broker and client. Job is allocated to the resource whose $Resource_{weightage}$ is immediate higher to $Job_{weightage}$ of submitted job. Broker sends job execution results to the client, after verification and validation of SQA.

Above discussed steps of proposed NDFS are depicted in Figure 1.

3.9. Enhancement of Efficiency of NDFS

This research also identifies that, in the instance of a resource failure resulting in possible deadline miss, instead of resubmitting the job, another optimal solution is possible. Partial backup is taken during execution of job, to ensure system becomes fault tolerant. Efficiency of NDFS is enhanced with addition of fault tolerance property. During execution of a job, if resource fails, possibility of meeting SQA will diminish drastically after job is resubmitted to another resource. To incorporate fault tolerance in NDFS, partial backup process is introduced with initialization fraction f, $0 < f < 1$, set at 0.3. For ensuring backup and seamless execution of job, state of the system of executing resource at 30%, 60% and 90% execution completion, with the job mirrored as well, are transmitted to next ranked resource. Figure 2 represents steps to be followed to embed the property of fault tolerance in NDFS.

Fault tolerant NDFS ensures, in case of resource failure, job execution gets resumed from last backed up point, therby saving on resubmission and execution time.

Implementation results of the algorithm, by executing several benchmark code instances in grid test bed, are presented in next section.

4. EXPERIMENTAL SETUP AND RESULTS

The real test bed of computational grid is set up using Globus Toolkit 5.2. The system parameter values are retrieved by SIGAR API through Java. The test bed consists of broker, 6 clients, and 5 resources.

Specifications of Client nodes: Dual Core Processor, 2 GB RAM and 160 GB HDD.

Specifications of Broker and Resources: Quad Core Processor, 4 GB RAM and 500 GB HDD.

4.1. Multiple Job Execution

Efficiency of an algorithm working in computational grid can only be certified if it can handle computation intensive jobs. To substantiate this, 3 instances of heterogeneous benchmark codes of Fast Fourier Transform (FFT) and another 3 instances of Matrix Multiplication are submitted to broker from 4 clients for possible execution in participating 5 resources in the grid.

The sequence of operations follows:

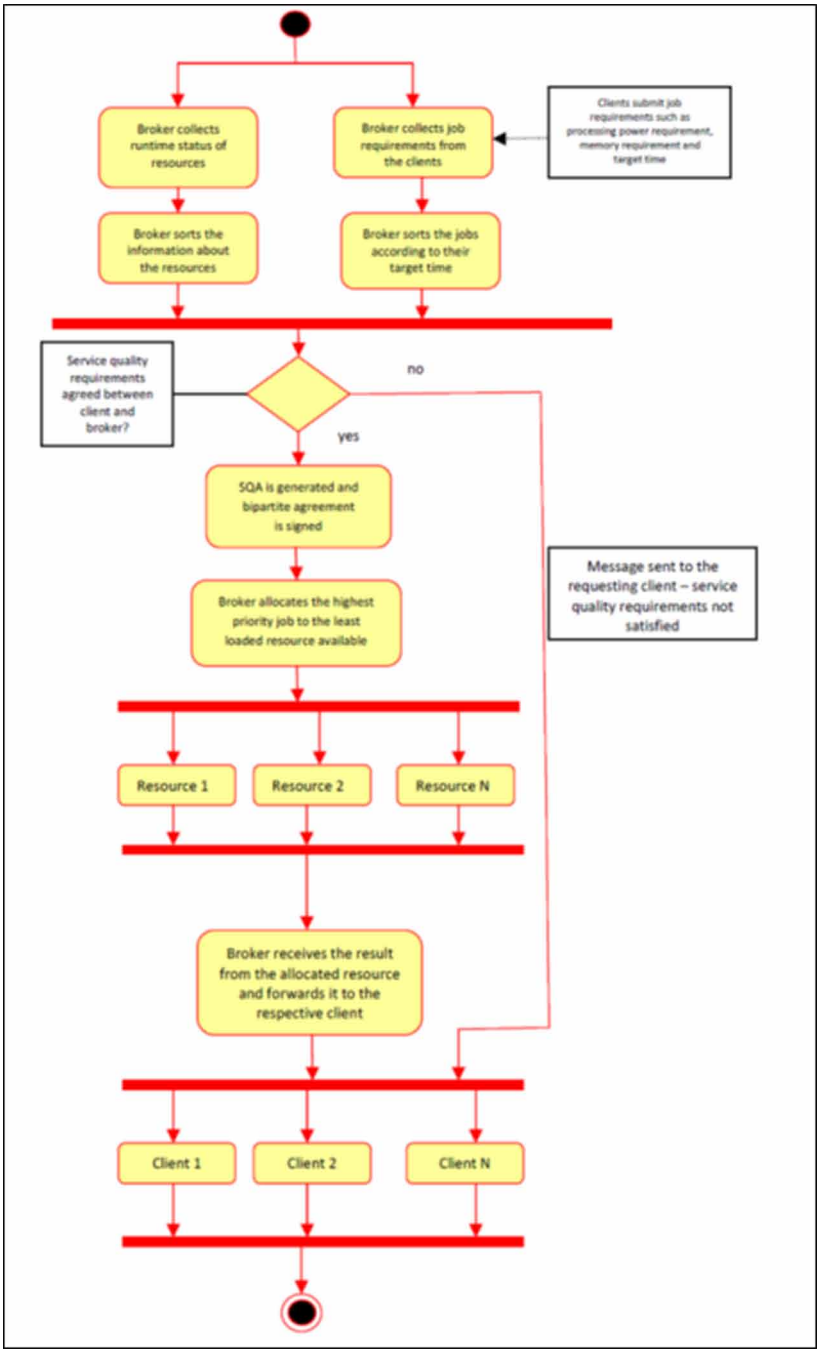
1. Client1 (192.168.30.3) sends "FFT1.java" job to Broker along with deadline specification.
2. System parameters are fetched from Client1 by Broker along with all resources in grid. $Job_{weightage}$ of submitted job and $Resource_{weightage}$ values of all participating resources are calculated by Broker.
3. By this time Client3 (192.168.30.9) sends "FFT2.java" job along with deadline specification to Broker.
   3.1 Again System parameters are fetched from Client3 by Broker along with all resources in grid. $Job_{weightage}$ of submitted job and $Resource_{weightage}$ values of all participating resources are calculated by Broker. These values are depicted in Table 1.
   3.2 SQAs are signed between Broker and both Client1 and Client3, as because matching values of $Job_{weightage}$ and $Resource_{weightage}$ have been found.
   3.3 Resource1 is assigned FFT1.java and Resource4 is assigned FFT2.java (presented in Table 1), as per the working principle of NDFS. SQAs are met.
4. Client2 (192.168.30.8) sends "MatMul1.java" job to Broker along with deadline specification.
   4.1 Resource5 is assigned MatMul1.java [presented in Table 1] by following the working principles of NDFS, as alrady discussed earlier. SQA is also met in this case as well.
5. Client4 (192.168.30.15) sends "MatMul2.java" job to Broker along with deadline specification.
   5.1 Resource3 is assigned MatMul2.java [presented in Table 1] by following the working principles of NDFS, as alrady discussed earlier. SQA is also met in this case.
6. Lastly, Client5 (192.168.30.16) and Client6 (192.168.30.17) send "FFT3.java" and "MatMul3.java" jobs to Broker. By virtue of working principle of NDFS, Resource1 and Resource2 are assigned these 2 jobs respectively (presented in Table 1). SQAs are met in these 2 cases as well.

Execution of fault tolerant NDFS version is discussed next.

4.2. Single Job Execution with Fault Tolerance

An instance of benchmark code of Matrix Multiplication, "MatMul.java", is executed in grid test bed by fault tolerant NDFS. Benchmark code is executed for four different cases, as depicted in Table 2.

Figure 1. Activity diagram of proposed NDFS



4.3. Case I

Job, "MatMul.java", is allocated to Resource2 to meet SQA as per non-fault tolerant NDFS, as represented in Figure 3. Execution is successful without resource failure, SQA is met.

Figure 2. Steps of fault tolerant NDFS

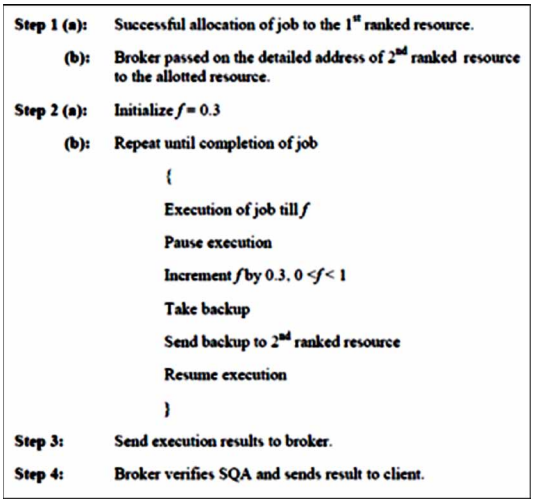


Table 1. Job Allocation by NDFS

Weightages	FFT1.java (Client1)	FFT2.java (Client3)	MatMul1.java (Client2)	MatMul2.java (Client4)	FFT3.java (Client5)	MatMul3.java (Client6)
$Job_{weightage}$	2.33	2.27	1.96	1.99	2.28	1.97
$Resource_{weightage}$ (After respective job submissions)	Resource1: 2.42 Resource2: 1.85 Resource3: 1.96 Resource4: 2.09 Resource5: 2.13	Resource1: 1.73 Resource2: 1.89 Resource3: 1.97 Resource4: 2.36 Resource5: 2.17	Resource1: 1.58 Resource2: 1.88 Resource3: 1.93 Resource4: 1.87 Resource5: 1.99	Resource1: 1.91 Resource2: 1.93 Resource3: 2.01 Resource4: 1.92 Resource5: 1.71	Resource1: 2.30 Resource2: 2.06 Resource3: 1.58 Resource4: 2.13 Resource5: 1.82	Resource1: 1.68 Resource2: 2.01 Resource3: 1.84 Resource4: 2.28 Resource5: 1.93

4.4. Case II

Job execution is halted after resource fails and job allocation is done again, resubmitted and then execution is successful. So, SQA compliance is not achieved because of deadline miss. Figure 3 represents this scenario of NDFS without fault tolerance.

Fault Tolerant NDFS solves this problem, case III and case IV describe that.

4.5. Case III

Client2 submits MatMul.java, the matrix multiplication job. Resource2, being the resource with highest rank, gets the allocation of the job by Fault Tolerant NDFS. Resource1 is next ranked resource. This case represents successful job execution, as depicted in TABLE 3.

- Ater job execution is 30% complete consuming 3.71 seconds, backup is taken in resource having 2nd rank, Resource1, by pausing execution at Resource2, in 0.61 seconds.

Table 2. Different cases for demonstrating fault tolerant NDFS

Case Type	Resource Scenario	Remarks
I	Active resources	Job execution is successful by Non-Fault Tolerant NDFS, SQA is met.
II	Resource fails	Job execution by Non-Fault Tolerant NDFS, after resource fails, job is resubmitted, SQA is not met.
III	Active resources	Job execution is successful by Fault Tolerant NDFS, SQA is met.
IV	Resource fails	Job execution is successful by Fault Tolerant NDFS, SQA is met.

- Same process is done after 60% and 90% completion of execution.
- Resource2 completes final 10% execution and Client2 receives the results via Broker.
- Though Fault Tolerant NDFS consumes more exection time compared to Non-Fault Tolerant NDFS, because of the backup approach, still SQA is met.

4.6. Case IV:

Client2 submits MatMul.java, the matrix multiplication job. It is allocated to Resource2, the highest ranked resource, by Fault Tolerant NDFS. Resource1 is next ranked resource. This case represents successful job execution inspite of resource failure.

- 30% execution is completed at Resource2 consuming 3.72 sec. Next, backup is sent to resource having 2nd rank, Resource1, by further consuming 0.6 sec.
- Resource2 fails after 43% completion of execution by consuming 1.62 sec more.
- Resumption of job execution happens from last back-up point in Resource1 (2nd ranked resource), and remaining execution (70%) is completed by Resource1 consuming 10.37 sec.
- SQA is complied.

Figure 3 presents this scenario.

Fault Tolerant NDFS ensures efficient performance in computational grid environment, as it has been observed in different scenarios that, SQA is complied in Cases I, III, IV and was not complied with in Case II.

4.7. Performance Comparison

Fault Tolerant NDFS is simulated with GridSim [4], the most widely used grid environment simulation tool. Performance of NDFS is compared with few algorithms, namely, Without Load Balancing (WLB), Fastest Processor to Largest Task First (FPLTF), Load Balancing on Enhanced GridSim (LBEGS), Max-Min, and Min-Min.

Tables 4-6 along with Figures 4-7 present comparison of the above mentioned algorithms with respect to various parameters such as, application execution time, communication overhead and rate of finished Gridlet. Processing entity number is constant at 200 and number of Gridlets is varied.

Figure 3. MatMul.java execution results for non-fault tolerant NDFS

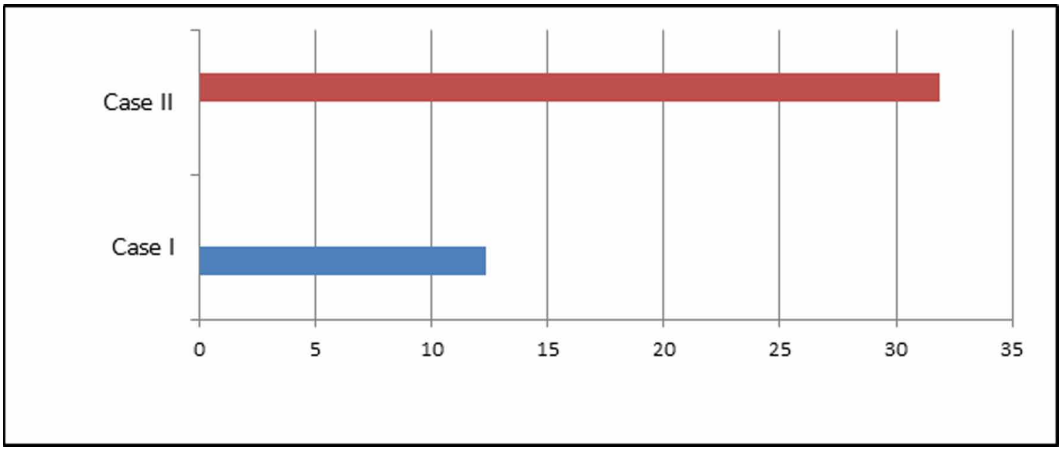


Table 3. Job completion timestamp values of fault tolerant NDFS

Fractional Co-efficient (f= 0.3)						
30%		60%		90%		Remaining (10%)
ECT	BCT	ECT	BCT	ECT	BCT	ECFT
3.71 sec	4.32 sec	8.16 sec	9.02 sec	12.78 sec	13.81 sec	15.06 sec

ECT : Execution Completion Timestamp
BCT : Backup Completion Timestamp
ECFT : Execution Completion Final Timestamp

5. CONCLUSION

Geographically dispersed resources constitute computational grid environment and these resources have largely varied workloads. In grid, number of job execution-deadline meets can be increased substantially by effective job scheduling among participating resources, also resulting in balanced workload across the grid. This target gets achieved in this research by introduction of average resource load along with current resource load, thereby improving the efficiency of NDFS algorithm. Service quality agreements are met by scheduling jobs in the adaptively ranked resources. Another difficult condition is encountered in computational grid environment when executing resource fails. This research deals with this problem by introducing partial backup approach by Fault Tolerant NDFS. The computational grid test bed is made by Globus Toolkit 5.2. Benchmark codes of multiple instances of Fast Fourier Transform and Matrix Multiplication are executed. Obtained results demonstrate balanced workload in computational grid along with fault tolerance property implemented. This research concentrates the implementation in Java and in future will be extended to .NET framework as well using Aneka software.

Figure 4. Job execution by fault tolerant NDFS inspite of resource failure

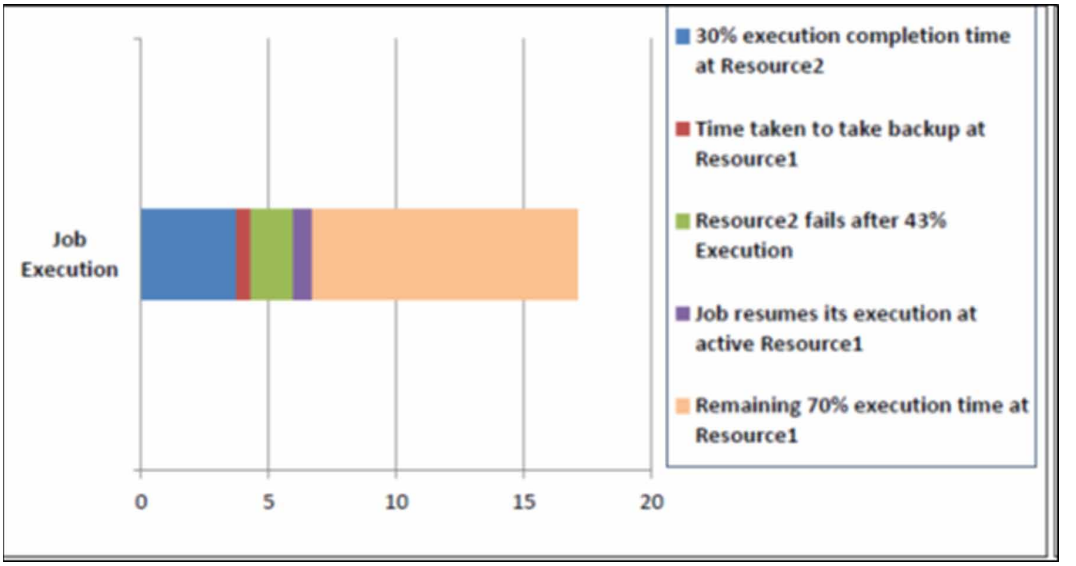


Table 4. No. of Communication Overheads comparison

Number of Gridlets	No. of Communication Overheads					
	WLB	LBEGS	Min-Min	Max-Min	FPLTF	Proposed NDFS
200	3	2	2	2	1	3
400	10	4	9	6	2	5
600	18	6	10	20	2	7
800	23	8	16	22	3	7
1000	30	11	28	24	4	7
1200	34	13	30	29	5	7
1400	39	14	36	32	5	8
1600	47	16	44	34	6	8
1800	52	18	47	43	6	8
2000	60	19	54	49	7	9

Table 5. Makespan comparison

Number of Gridlets	Makespan (in seconds)					
	WLB	LBEGS	Min-Min	Max-Min	FPLTF	Proposed NDFS
200	2.2	1.7	2.1	1.7	1.6	1.4
400	3.1	2.3	3.8	2.4	2.2	2
600	4	3.8	4.2	3.9	3.6	3.3
800	5.3	5.6	5.8	5.7	5	4.7
1000	7.1	6.4	7.6	6.8	5.8	5.4
1200	8.1	8.2	8.2	7.8	6.6	6.1
1400	10.2	10.2	10	9	7.6	7.1
1600	12.3	11.9	11.6	10.8	9	8
1800	13.8	13.2	12.7	11.8	10.5	9.3
2000	15.5	15	14.2	13.9	12.2	10.5

Table 6. Comparison of No. of finished Gridlets

Number of Gridlets	Number of Finished Gridlets					
	WLB	LBEGS	Min-Min	Max-Min	FPLTF	Proposed NDFS
200	160	200	170	180	200	200
400	330	383	341	356	391	397
600	504	530	545	562	580	589
800	615	620	662	720	753	770
1000	671	685	754	798	887	930
1200	785	816	905	988	1075	1115
1400	926	961	1055	1147	1235	1290
1600	1060	1105	1217	1311	1402	1465
1800	1195	1281	1396	1492	1591	1650
2000	1325	1417	1545	1638	1755	1833

Figure 5. Communication overhead comparison, PEs=200

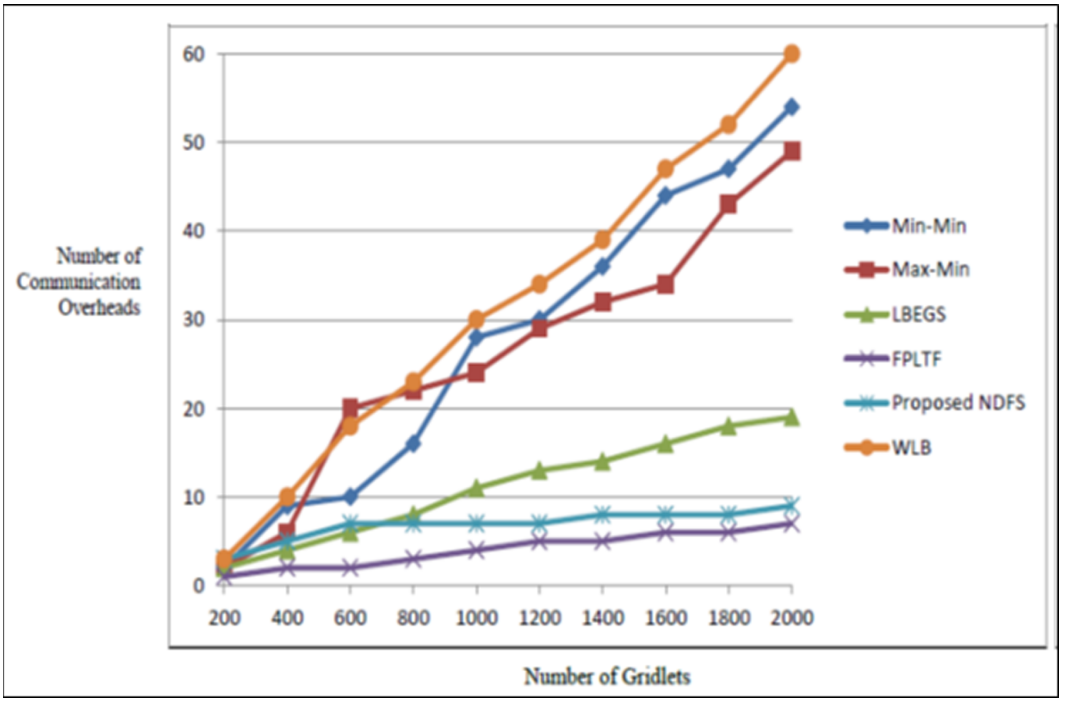


Figure 6. Makespan comparison, PEs=200

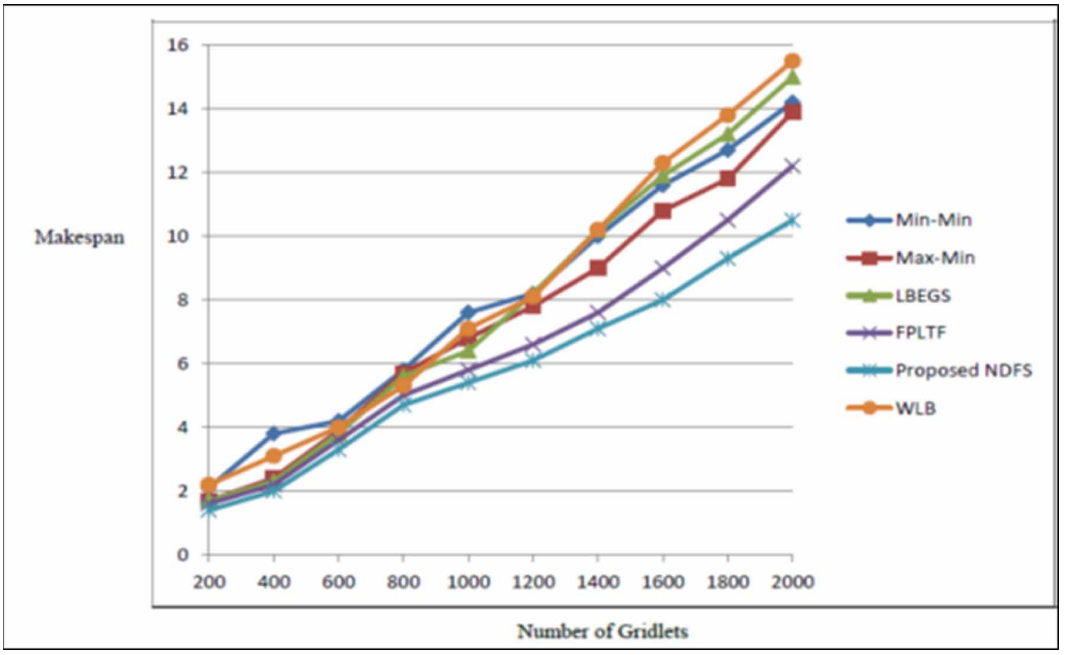
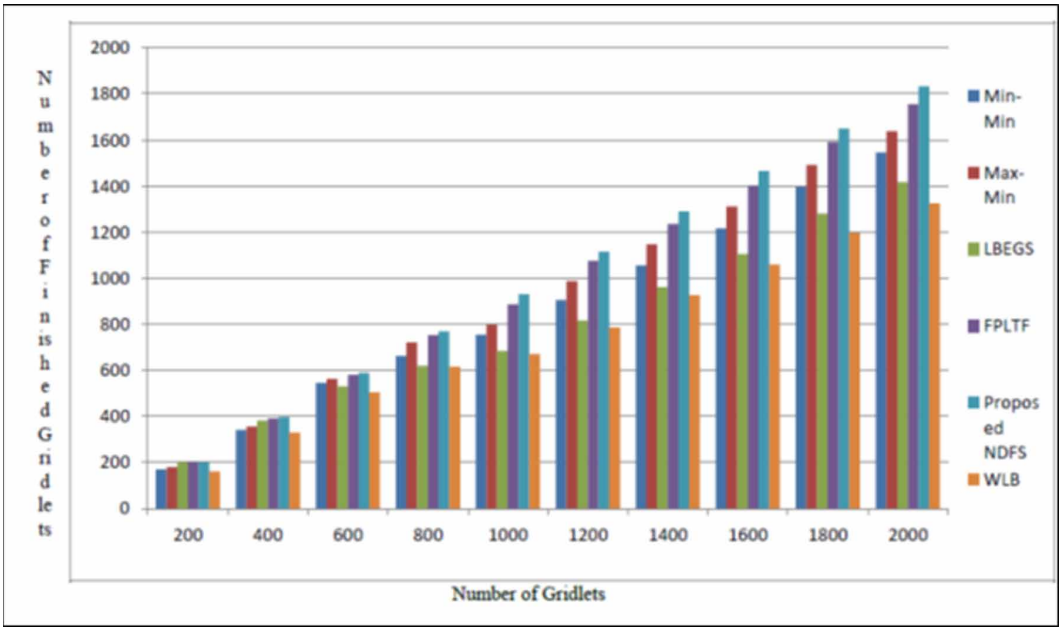


Figure 7. No. of finished gridlets comparison, PEs=200



REFERENCES

Abo Rizka, M., & Rekaby, A. (2012). Dynamic Job Scheduling and Load balancing algorithm In Grid Environment via X-Dimension binary tree data model. *International Journal of Intelligent Computing & Information Science*, *12*(2).

Balasangameshwara, J., & Raju, N. (2012). A hybrid policy for fault tolerant load balancing in grid computing environments. *Journal of Network and Computer Applications*, *35*(1), 412–422. doi:10.1016/j.jnca.2011.09.005

Balasangameshwara, J., & Raju, N. (2013). Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost. *IEEE Transactions on Computers*, *62*(5), 990–1003.

Buyya, R., & Murshed, M. (2002). GridSim: A toolkit for the modeling and simulation of distributed management and scheduling for Grid computing. *The Journal of Concurrency and Computation: Practice and Experience*, *14*(13).

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. doi:10.1007/s10723-009-9120-9

Di, S., Kondo, D., & Cirne, W. (2014). Google hostload prediction based on Bayesian model with optimized feature combination. *Journal of Parallel and Distributed Computing*, *74*(1), 1820–1832. doi:10.1016/j.jpdc.2013.10.001

Foster, I., Kesselman, C., & Tuccke, S. (2001). The Anatomy of the Grid. *The International Journal of Supercomputer Applications*, *15*(3), 200–222. doi:10.1177/109434200101500302

Goswami, S., & Das, A. (2014). Handling Resource Failure towards Load Balancing in Computational Grid Environment. In *Proceedings of the Fourth International Conference on Emerging Applications of Information Technology* (pp 133-138). IEEE. doi:10.1109/EAIT.2014.62

Goswami, S., & Das, A. (2015). Deadline Stringency Based Job Scheduling in Computational Grid Environment. In *Proceedings of the International Conference on Computing for Sustainable Global Development, 9th IndiaCom - 2015* (pp 531-536). IEEE.

Goswami, S., & Das, A. (2015). Resource Prioritization Technique in Computational Grid Environment. In *Proceedings of the Second International Conference on Computer and Communication Technologies - IC3T 2015* (pp 765-772). Springer.

Goswami, S., & Das, A. (2016). An Adaptive Resource Allocation Scheme in Computational Grid. *International Journal of Control Theory and Applications*, *9*(41), 721–736.

Goswami, S., & Das, A. (2016). Optimisation of Workload Scheduling in Computational Grid. In *Proceedings of the FICTA-2016*. Singapore: Springer.

Jaiswal, S., Mishra, A., & Bhanodia, P. (2014). Grid host load prediction using gridsim simulation and hidden markov model. *International Journal of Emerging Technology and Advanced Engineering*, *4*(7), 775–781.

Kant Soni, V., Sharma, R., & Kumar Mishra, M. (2010). An analysis of various job scheduling strategies in grid computing. In *Proceedings of 2nd International Conference on Signal Processing Systems (ICSPS) (Vol. 2, pp. 162). IEEE.*

Karthick Kumar, U. (2011). A Dynamic Load Balancing Algorithm in Computational Grid Using Fair Scheduling. *International Journal of Computer Science Issues*, *8*(5), 123–129.

Keerthika, P., & Kasthuri, N. (2013). A hybrid scheduling algorithm with load balancing for computational grid. *International Journal of Advanced Science and Technology*, *58*, 13–28. doi:10.14257/ijast.2013.58.02

Rajavel, R. (2010). De-Centralized Load Balancing for the Computational Grid Environment. In *Proceedings of International Conference on Communication and Computational Intelligence*. IEEE.

Ray, S., & De Sarkar, A. (2013). Resource Allocation Scheme in Cloud Infrastructure. In *Proceedings of International Conference on Cloud & Ubiquitous Computing & Emerging Technologies*. doi:10.1109/CUBE.2013.16

Ruchir, S., Bharadwaj, V., & Manoj, M. (2007). On the design of adaptive and de-centralized load balancing algorithms with load estimation for computational grid environments. *IEEE Transactions on Parallel and Distributed Systems*, *18*(12).

Saaty, T. L. (2008). Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, *1*(1), 83–98. doi:10.1504/IJSSCI.2008.017590

Shestak, V., Smith, J., Siegel, J. H., & Maciejewski, A. A. (2008). Stochastic robustness metric and its use for static resource allocations. *Journal of Parallel and Distributed Computing*, *68*(8), 1157–1173. doi:10.1016/j.jpdc.2008.01.002

Stal, M. (1995). The Broker Architectural Framework. In *Proceedings of the Object-Oriented Programming Systems, Languages and Applications Conference (OOP SLA'95).*

Sukalyan Goswami has received his M.Tech in Information Technology from IIIT-Bangalore in 2006 and his B.E. in Electronics & Communication Engineering from VTU, Karnataka in 2004. He has eleven years of teaching experience and one and a half years of industry experience. He has twelve international conference / journal publications. His research interest is grid computing. He is currently engaged with the Institute of Engineering & Management, Kolkata, as an Assistant Professor in the Department of Computer Science & Engineering.

Kuntal Mukherjee has done PhD in computer science. He has vast experience of teaching & research and has several international conference / journal publications. His research interest is grid computing, quantum computing and cloud computing. He is currently engaged with Birla Institute of Technology, Mesra, Lalpur Campus, in the Department of Computer Science & Engineering.