ON FAULT TOLERANCE OF RESOURCES IN COMPUTATIONAL GRIDS

Arindam Das[1] and Ajanta De Sarkar[2]

[1]Department of MCA, MCKV Institute of Engineering, Howrah, India
arindamdas.mckv@gmail.com
[2] Department of Computer Science & Engineering, Birla Institute of Technology, Mesra, Kolkata Campus, Kolkata 700 107
adsarkar@bitmesra.ac.in

ABSTRACT

Grid computing or computational grid is always a vast research field in academic, as well as in industry also. Computational grid provides resource sharing through multi-institutional virtual organizations for dynamic problem solving. Various heterogeneous resources of different administrative domain are virtually distributed through different network in computational grids. Thus any type of failure can occur at any point of time and job running in grid environment might fail. Hence fault tolerance is an important and challenging issue in grid computing as the dependability of individual grid resources may not be guaranteed. In order to make computational grids more effective and reliable fault tolerant system is necessary. The objective of this paper is to review different existing fault tolerance techniques applicable in grid computing. This paper presents state of the art of various fault tolerance technique and comparative study of the existing algorithms.

KEYWORDS

Component, Computational grids, Fault tolerance, Failure, Checkpointing

1. INTRODUCTION

Grid is an association of computer resources from several administrative domains to reach a mutual goal with an abstraction of service origination to the user. This view is often put to as an analogy to power grids where consumers get access to electricity through sockets on wall with no care for where and how that electricity is generated. Similarly, in computational grid the users can access any resources like, process, storage, data and applications with little or no knowledge of physical locations of those resources and the underlying technologies used.

Grid computing is special because here the nodes can be purchased as commodity hardware. These nodes are easily combined to produce a similar computing resource like multiprocessor supercomputer but at a lower cost. The large number of processors to be managed and lack of high-speed connections in grid can be complemented well by multiple independent parallel computations. Grid computing involves the aggregation of large-scale cluster computing based systems.

As resources in grid are used outside of organizational boundaries, it becomes increasingly difficult to guarantee that a resource being used is not malicious in some way or failure of resources is uncertain. Fault tolerance in grid computing is necessary to preserve the delivery of

DOI: 10.5121/ijgca.2012.3301

expected services despite the presence of fault caused errors within the system itself. It aims at the avoidance of failures in the presence of faults, as availability of grid resources is dynamic.

The objective of this paper is to review a few fault tolerant techniques in grid computing. This paper presents state of the art of various existing fault tolerance technique and comparative study of the fault tolerant algorithms. Brief overview of grid computing is discussed in Section 2. Section 3 explains faults and failure and concepts of fault tolerance in grid computing. Review of a few algorithms related to fault tolerance is explained in Section 4. Section 5 concludes the paper with directives of future work.

2. GRID COMPUTING

A grid [3], known to be a large-scale virtual organization, is enabling to solve complex scientific and compute-intensive problems. The virtual organization is formed with geographically distributed hardware and software infrastructure of flexible, secure and coordinated shared vast amounts of heterogeneous resources from multiple administrative domains. Computational grid environment is shown in Figure 1. Heterogeneous computational nodes have connected to form a Grid test-bed. In this test-bed registered resource database and Grid resources server is also shown. Server or database might be accessed during computation of large job. User can submit job through any node among Node A, Node B, Node C, Node D or Node E in Grid. Job might necessitate adapting the changed resource scenario in Grid environment. Hence, fault tolerance of resources is major challenging issue in dynamic virtual computational Grids.

Considering multiple administrative domains owned by multiple individuals/organizations the intension of participation or volunteering might not always be trustworthy. Besides, the duration of participation might also not be consistent. As in grid, all the resources are connected through heterogeneous network. There might be no guarantee that the nodes would not be dropped out of the network at random times. The impacts of trust and availability on performance and development difficulty can influence the choice of whether to deploy on to a dedicated computer cluster in the developing organizations or to an open external network of contractors.

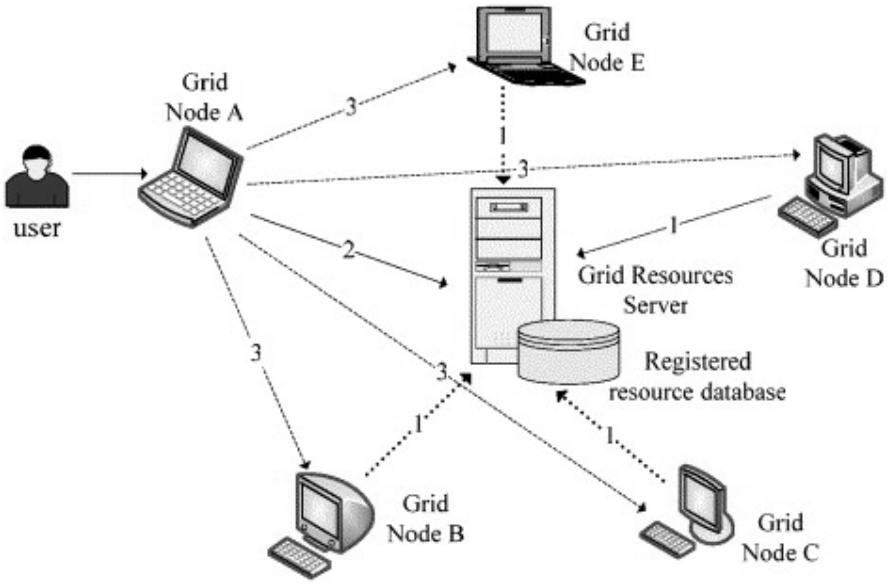

Figure 1 : Computational Grid Environment [15]

On the other side, with many languages in place, there has been a tradeoff between investment in software development and the number of platforms that can be supported. Cross-platform languages can reduce the need to make this tradeoff. Various middleware based projects have created genuine infrastructure to allow diverse scientific and commercial projects to harness a particular associated grid or for the purpose of setting up new grids. Middleware can be seen as a layer between the hardware and software. SLA management, License management, portals and data management are major research issues in grid middleware.

Due to unavailability of network or development difficulty or faulty resources, fault may occur in the results or performance may be degraded. Fault tolerance is the ability to preserve the delivery of expected services despite the presence of fault caused errors within the system itself. It aims at the avoidance of failures in the presence of faults. A fault tolerant service detects errors and recovers them without participation of any external agents, such as humans. Errors are detected and corrected and permanent faults are located and removed while the system continues to deliver acceptable services.

Faults and failure in grid are explained in the next section.

3. FAULTS AND FAILURES IN GRID

Fault tolerance is an important property in grid computing as the dependability of individual grid resources may not be guaranteed. In many cases, an organization may send out jobs for remote execution on resources upon which no trust can be placed; for example, the resources may be outside of its organizational boundaries, or may be shared by different users at the same time. A fault tolerant approach may therefore be useful in order to potentially prevent a malicious node affecting the overall performance of the application. As applications scale to take advantage of Grid resources, their size and complexity will increase dramatically.

A major challenge in a dynamic grid with thousands of nodes connected to each other is fault tolerance. The more resources and components involved the more complicated and error-prone becomes the system. To comprehend fault tolerance mechanisms, it is important to point out the difference between faults, errors and failures.
Fault: A fault is a violation of a system's underlying assumptions.
Error: An error is an internal data state that reflects a fault.
Failure: A failure is an externally visible deviation from specifications.

In reality, a fault need not result in an error, or an error in a failure.
Different types of faults, classified based on several factors, are mentioned in the following:

- *Physical faults:* faulty storage, faulty CPUs, faulty memory.
- *Unconditional termination:* Mostly, user pressed Ctrl+c.
- *Network faults:* packet corruption, faults due to network partition, packet loss.
- *Lifecycle faults:* Legacy or versioning faults.
- *Processor faults:* Machine or operating system crashes.
- *Media faults:* Disk head crashes.
- *Service expiry fault:* The service time of a resource may expire while application is using the resources in grid.
- *Process faults:* software bug, resource shortage.
- *Interaction faults:* timing overhead, protocol incompatibilities, security incompatibilities, policy problems.

A failure, irrespective of its cause (either error or fault), is observed or reported first in a grid system. Three types of failures such as, Permanent, Intermittent and Transient, which can occur in computer systems with respect to time. Due to these failures system may behave in different ways. Three types of behaviors are possible in systems after a failure:

- *Failstop system:* The system does not output any data once it has failed. It immediately stops sending any events or messages and does not respond to any messages.
- *Failfast system:* The system behaves like a Byzantine system for some time but moves into a *failstop mode* after a short period of time. It does not matter what type of fault or failure has caused this behavior but it is necessary that the system does not perform any operation once it has failed.
- *Byzantine system:* The system does not stop after a failure, instead behaves in a inconsistent way. It may send out wrong results of the application.

4. REVIEW OF FAULT TOLERANCE TECHNIQUES

The main objective of grid computing is to maintain the workflows or services in presence of faults so that no failure stage is reached. This section presents a few fault tolerant techniques in the following:

4.1. Job and Data Replication

The term, replication implies making copies or replicas of an existing entity. In grid environment, job/task or data are replicated to tackle the faults. Workflows can be executed with a rule-based system with a framework operating dynamically taking an appropriate fault tolerance technique. The framework considers [2] the user's preferences and the grid resource allocation situation to decide, using a binary tree, the most appropriate fault tolerant technique to be used for each task that can compose a workflow.

Besides, most of the existing replication-based algorithms use a fixed number of replications for each job which consumes more grid resources. To overcome this, Adaptive Job Replication (AJR) algorithm is proposed to adaptively determine the number of job replicas [10] according to the grid failure history followed by Backup Resources Selection (BRS) algorithm to schedule these replicas.

Moreover, Fault Tolerance using Adaptive Replication in Grid Computing (FTARG) [14] is an adaptive replication middleware which addresses the fault tolerance of grid based applications by providing data replication at different sites. FTARG is an Aneka based grid middleware designed for high-performance grid based applications. FTARG enables data synchronization between multiple heterogeneous databases located in the grid by supporting a variety of synchronization modes. However, it can delay the job execution if the number of replicas [11] to be generated is not reasonably determined for data or task replication.

Resubmission Impact, a heuristic, for fault tolerance in workflow executions [8] is a new technique applicable for distributed environment. This heuristic is based on a combination of task replication and task resubmission using a resubmission impact metric which measures the impact of repeated task resubmission on the execution time of a workflow. The metric used in the heuristic is to define the number of replications generated for each workflow task. In contrast to related approaches, this method can be used effectively on systems even in the absence of historic failure trace data.

So, in a replication based fault recovery, availability of replicated data at different sites of computing increases the fault tolerance adaptability. Data synchronization among cross vendor databases should also be addressed. In case of job/task replication, heuristic method based on task resubmission impact measures can be better than the traditional approach with little or no dependency on failure of historic data. However, in both data and job replication, determination of the appropriate number of replicas plays a significant role.

4.2. Checkpointing

Checkpointing [14] is the process of saving the state of an application in execution to a stable storage for future utilization. In case of any fault, this saved state is to be referred to resume execution of the application from the point in the computation where it was last checkpointed.

In addition to the traditional fault tolerance techniques, specific checkpoint-recovery schemes are needed in grid workflow management systems of present time to address the new reliability challenges. A Fault Tolerance and Recovery component that extends the ActiveBPEL workflow engine [2] has been proposed to develop mechanisms for building an autonomic workflow management system that effectively detects, diagnoses, notifies, reacts and recovers automatically from failures during workflow execution. The detection mechanism inspects the messages exchanged between the workflow and the synchronized Web Service components for the search of faults. The default behavior of ActiveBPEL can be modified in order to recover a process from a faulty state, using a non-intrusive checkpointing mechanism.

Further, a new strategy named Resource Fault Occurrence History (RFOH) [9] for fault tolerant job scheduling in computational grid is proposed. This strategy maintains the history of fault occurrence of resources in Grid Information Server (GIS). A resource broker with jobs to schedule, it uses this GIS information in Genetic Algorithm and looks for a near optimal solution for the problem. Next, it raises the percentage of jobs executed within specified deadline. Using checkpoint techniques, the proposed strategy can make grid scheduling more reliable and efficient.

A proposition is also there to present an experience to endow with fault tolerance support parallel executions on grids through the integration of ComPiler for Portable Checkpointing (CPPC) [1], a checkpointing tool for parallel applications, and GridWay: a meta-scheduler provided with the Globus Toolkit. One of the strengths of this proposal is, transparency and ease of use i.e. the CPPC-GW infrastructure will take care of all viz. automatically submitting and monitoring the application, making remote backups of checkpoint files, detecting faults and migrating and restarting failed executions.

Periodic job checkpointing, being very robust, it can detain job execution if checkpointing intervals are inappropriately chosen. Moreover, fault tolerance poses an important problem in the scope of grid computing environments. The heuristics have been evaluated in Dynamic Scheduling in Distributed Environments (DSiDE) grid simulator under changing system load and availability. In [11] the results have shown that the runtime overhead characteristic to periodic checkpointing can significantly be reduced when the checkpointing frequency is dynamically adapted in function of resource stability and remaining job execution time.

Reducing the number of crashed node can cut down the number of faulty tasks by efficient node allocation. To improve fault tolerance and decrease price cost of a job with an acceptable completion time [1], a predictive method to select the best nodes and sites is proposed.

4.3. Scheduling/Agent based migration

To overcome the drawbacks present with checkpointing and replication mechanisms, fault tolerance is factored into grid scheduling. Scheduling policies for grid systems can be classified into space sharing and time sharing [14] policies. It is also possible to combine these two types of policies into a hybrid policy.

Security-aware and fault-tolerant job scheduling is crucial to achieve high performance in an open grid computing environment. On the contrary, the fixed fault-tolerant strategy in job scheduling may improperly exploit excessive resources. A cloud model is proposed [13] to manage the fuzziness and uncertainties of task scheduling while deciding the kind of fault-tolerance strategy to be selected, at the same time, for each individual job to ensure more reliability of computation and shorter makespan.

In fault-tolerant scheduling, primary-backup approach is a practiced methodology used for fault tolerance where each task holds a primary copy and a backup copy submitted to two different processors. For independent tasks, a backup copy can be overloaded with other backup copies on the same processor while for dependent tasks, precedence constraint among tasks must be considered during scheduling of backup copies and overloading backups. Two algorithms: the Minimum Replication Cost with Early Completion Time (MRC-ECT) algorithm and the Minimum Completion Time with Less Replication Cost (MCT-LRC) algorithm has been proposed to schedule backups of independent jobs and dependent jobs, respectively [13]. Algorithm MRC-ECT is observed guarantee an optimal backup schedule in terms of replication cost for an independent task, while MCT-LRC can schedule a backup of a dependent task with minimum completion time and less replication cost.

4.4. Load balancing

In general, the P2P system has similar objective to that of the grid system since both of them coordinate the large sets of distributed resources. Therefore, many projects keep on integrating these two complementary technologies i.e. *P2P grid* to perform as an ideal distributed computing system.

Besides, despite many load-balancing approaches has been proposed for real-time applications in parallel and distributed systems, there are very less work found on the impact of fault tolerance policies for load balancing mechanisms. A fault tolerant policy has been proposed to balance loads dynamically in the P2P grid system, named the Fault Tolerant policy on Dynamic Load Balancing (FTDLB) [17]. In order to minimize the job turnaround time, mostly, is the primary the goal of load balancing and FTDLB could adaptively adjust the load of real-time applications to achieve the job's minimal turnaround time.

FTDLB policy can tolerate the node's permanent failures while balancing load of real-time applications on P2P grids. It calculates the job turnaround time in each node and dynamically allocates job to the befitting node for execution as per the evaluation. The existing job migration time estimation has been improved here. The policy can tolerate the node's permanent failures while balancing load of real-time applications on P2P grids. For improving the system reliability, FTDLB duplicates jobs (copy approach) into different sites to tolerate failures by keeping a high availability rate at the increase of error rate. Experimental results reveal that FTDLB policy indeed improves the job completion rate.

Load balancing strategies are classified as *dynamic* and *static*. In general, the static load balancing strategy [19] needs the prior information to make decisions, such as the execution rate of each node, for load distribution.

On the other hand, the dynamic load balancing strategy [6] exploits the system information to make decisions at run time. Load balancing strategies could also be categorised as centralized or decentralized [12]. The centralized strategy selects a single processor to handle load scheduling, while the distributed strategy welcomes each participating node to handle load balancing.

4.5. Global behaviour modelling

One of the most confusing aspects of grid systems is that theoretically they are considered as single elements but, when it comes to practice, especially in management related issues, they are considered as a set of independent, loosely related elements.

To illustrate this idea, it is interesting first to analyze the case of a single desktop computer. It apparently looks a much simpler system and is commonly regarded as a single device but, in fact, it is composed of a large set of sophisticated elements that cooperate themselves. Elements like CPUs, memory and its controllers, video cards, hard drives; network interfaces etc. have distinctive functionalities and are technologically complex. Still, they are seen as parts of a single entity, instead of a set of heterogeneous resources. This change of perspective is due to the use of high-level software tools (basically the Operating System) that provide an abstraction layer between the real, heterogeneous and complex hardware range and the user. Several generic parameters are defined, such as CPU load or network usage, in order to express the system state in a standard manner. Even though this abstraction carries some loss of information, it allows the managing techniques to be standardized, regarding all desktop computers by the same parameters. Considering fault tolerance, generic procedures are developed in the same way.

Distinguished by its point of view, fault tolerance techniques in grid systems can be split into two categories: *resource-level* (focused on every machine) and *service-level* (focused on *global behavior*). In order to optimize performance and increase system dependability the correct combination of these two types of techniques should be applied.

However, some important aspects must be considered. The *resource-level* fault tolerance involves the application of standard fault tolerance techniques in each and every one of the resources in the system. This might seem very straightforward, but careful consideration reveals that most of typical grid characteristics could limit its efficiency. The heterogeneous and non-dedicated nature of the system increase complexity, but it is the non-centralized aspect the one that becomes the great difficulty.

In many cases, the global management system has so limited control of each resource that the only suitable solution seems to increase redundancy.
The *service-level* fault tolerance, on the other hand, deals with system-wide policies aiming to increase dependability of the services provided. This is particularly important in utility computing systems, where the quality-of-service (QoS) is the key factor. However, as the fault tolerance policies have to deal with the whole system, it is important to find ways to efficiently manage this complexity. It is also important to understand that, as the nature of the system is different from resource-level fault tolerance, the terms in which this fault tolerance is expressed certainly differ.

In *resource-level* fault tolerance basic concepts such as fault or failure are directly inherited from traditional distributed systems. Events such as a machine turning unexpectedly off or the temporary loss of a network link are clearly regarded as faults. But in a non-dedicated, non-centralized distributed system like a grid, each partner that shares resources keeps full control over its property (computing nodes, storage elements, network links, etc). Resource providers can change the state of its own resources, without consent from the grid global management. For instance, some machines could be turned off, originating an event that would be probably

considered a fault in traditional distributed systems fault tolerance. But in grid systems these events are by no means considered as undesirable or unexpected. They are more likely accepted situations that not only may, but will occur as part of the natural evolution of the grid. Therefore, *service-level* fault tolerance can never regard them as faults.

The *service-level* fault tolerance should focus on QoS issues and *global behavior*. It can benefit from a representation of the grid global state in a service oriented form. This would become a behavior model based on globally service-relevant states instead of the multiple specifics of each resource. This representation not only seems ideal for service-level fault tolerance, but also provides the abstraction layer mentioned in the previous subsection. With such a model, grid management tools could finally have the previously mentioned single entity perspective, incorporating the system's complexity without being overwhelmed by it. This could also take *service-level* fault tolerance a step further, better understanding and improving the systems behaviour and dependability.

5. CONCLUSION AND FUTURE WORK

This paper presents a comparative survey of fault tolerance in grid environment. The accepted techniques of fault-tolerance in grid environment with their importance, combinations and variations have been discussed.Replication of job/data is necessary in order to increase the resource availability to the computing nodes. An application of checkpointing is important to formulate organized policies to recover from a system under errors or faults. It effectively prevents system from being led to a failure state. Security aware scheduling of grid jobs migrated through agents improves grid performance significantly. A newer concept, service level behavior or *global behavior* on understanding the grid services representing a one unified service system has changed the whole understanding perspective of fault scenario in grid computing. Additionally, grid resource management in terms of deploying load balancing techniques enhances grid performance, too.

However, accepting the importance of all the aforesaid areas, to put forward a future direction of work, this research would next focus on checkpointing technique for better performance of applications running in grid. It would address the following issues as next course of work:

- *Checkpoint fixation level:* either at system level (i.e. at OS or middleware level) or at application level.

- *In-transit and Orphan message management with checkpoint:* latency and resources held up for this reason would be freed if applied with a suitable policy.

- *Scope of Checkpoint: local* – for each process instance or *global* – for each parallel program in execution.

- *Storage space requirement for checkpointing: light* – only the first/top level assignment is stored thereby less storage and communication overhead and *heavy* – in addition to light, newly learnt clauses saved atop the decision stack.

- *Granularity of checkpointing: full* – entire state of application saved and *incremental* – application state saved from previous checkpoints only.

REFERENCES

[1] Asgarali Bouyer, Abdul Hanan Abdullah, Hasan Ebrahimpour and Firouz Nasrollahi," Fault-Tolerance Scheduling by Using Rough Set based Multi-Checkpointing on Economic Grids", International Conference on Computational Science and Engineering, 2009, IEEE Computer Society, pp.103-109.

[2] Elvin Sindrilaru, Alexandru Costan and Valentin Cristea, "Fault Tolerance and Recovery in Grid Workflow Management Systems," International Conference on Complex, Intelligent and Software Intensive Systems, 2010, IEEE Computer Society, pp.475-480.

[3] Felipe Pontes Guimaraes and Alba Cristina Magalhaes Alves de Melo, "User-Defined Adaptive Fault-Tolerant Execution of Workflows in the Grid," 11th IEEE International Conference on Computer and Information Technology, 2011, IEEE Computer Society, pp.356-362.

[4] Ian Foster, "What is the Grid? A Three Point Checklist", Argonne National Laboratory & University of Chicago, July 20, 2002.

[5] Ivan Cores, Gabriel Rodriguez, Maria J. Martin and Patricia Gonzalez,"Achieving Fault Tolerance on Grids with the CPPC Framework and the GridWay Metascheduler", 22nd International Symposium on Computer Architecture and High Performance Computing, 2010, IEEE Computer Society, pp.119-126.

[6] J. Chen, B. Lu, "Load Balancing Oriented Economic Grid Resource Scheduling", IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, 2008, pp.813-817.

[7] Jesus Montes, Alberto Sanchez and Maria S. Perez, "Improving grid fault tolerance by means of global behavior modeling", Ninth International Symposium on Parallel and Distributed Computing, 2010, IEEE Computer Society, pp.101-108.

[8] Kassian Plankensteiner, Radu Prodan and Thomas Fahringer, "A New Fault Tolerance Heuristic for Scientific Workflows in Highly Distributed Environments based on Resubmission Impact," Fifth IEEE International Conference on e-Science,2009, IEEE Computer Society, pp.313-320.

[9] Leili Mohammad Khanli,Maryam Etminan Far,Amir Masoud Rahmani, "RFOH: A New Fault Tolerant Job Scheduler in Grid Computing," Second International Conference on Computer Engineering and Applications, 2010, IEEE Computer Society, pp.422-425.

[10] M. Amoon, "Design of a Fault-Tolerant Scheduling System for Grid Computing," Second International Conference on Networking and Distributed Computing, 2011, IEEE Computer Society, pp. 104-108.

[11] Maria Chtepen, Filip H.A. Claeys, Bart Dhoedt, Filip De Turck, Piet Demeester and Peter A. Vanrolleghem, "Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 20, NO. 2, FEBRUARY 2009. IEEE Computer Society, pp.180-190.

[12] P. Krueger, M. Livny, "A Comparison of Preemptive and Non-Preemptive Load Balancing," 8th International Conference on Distributes Computing Systems, Jun 1988, pp 123-130.

[13] Qin Zheng, Bharadwaj Veeravalli and Chen-Khong Tham, "On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs," IEEE TRANSACTIONS ON COMPUTERS, VOL. 58, NO. 3, MARCH 2009, IEEE Computer Society, pp.380-393.

[14] S. Siva Sathya, K. Syam Babu, "Survey of fault tolerant techniques for grid," ELSEVIER ScienceDirect, COMPUTER SCIENCE REVIEW 4(2010) pp.101 -120.

[15] Shen et al, “System design and implementation of digital-image processing using computational grids”, Computers & Geosciences, volume 31, Issue 5, pp: 619-630

[16] Srinivasa K G, Siddesh G M and Sijo Cherian, "Fault-Tolerant middleware for Grid Computing," 12th IEEE International Conference on High Performance Computing and Communications, 2010, IEEE Computer Society, pp.635-640.

[17] Tian-Liang Huang, Tian-An Hsieh, Kuan-Chou Lai, Kuan-Ching Li, Ching-Hsien Hsu and Hsi-Ya Chang,"Fault Tolerance Policy on Dynamic Load Balancing in P2P Grids," International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11, 2011, IEEE Computer Society, pp.1413-1420.

[18] Yi Hu, Bin Gong and Fengyu Wang, "Cloud Model-Based Security-aware and Fault-Tolerant Job Scheduling for Computing Grid,",The Fifth Annual ChinaGrid Conference, IEEE Computer Society, pp.25-30.

[19] Y. Pan, W. Lu, Y. Zhang, K. Chiu, “A Static Load-Balancing Scheme for Parallel XML Parsing on Multicore CPUs”, Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07), 2007, pp.351-362.

Authors

Arindam Das is presently working as Assistant Professor in MCKV Institute of Engineering, Howrah, India. After graduating in Science from Calcutta University in 1997, he started his career as Laboratory Instructor in a leading Engineering Institution. Later, he completed his MCA from Indira Gandhi National Open University in 2004. He completed his MTech in Information Technology from Bengal Engineering and Science University in 2009. Currently he is pursuing PhD in the field of Grid Computing.

Ajanta De Sarkar is working as Associate Professor in the department of CSE in Birla Institute of Technology, Mesra. She is having altogether 16 years of experience including 6 years of Industry experience. Having graduated from Bethune College, University of Calcutta in B.Sc. (Mathematics) in 1993, and obtained MCA degree in 1996 from Jadavpur University. She has been awarded PhD in Engg. from Jadavpur University in 2009. Her field of Specialization is Distributed Computing, specifically Grid Computing. Her focused research area includes Grid Computing, Cloud Computing and Wireless Sensor Network.