

Fault TOLERANCE IN GRID COMPUTING: STATE OF THE ART AND OPEN ISSUES

Ritu Garg and Awadhesh Kumar Singh

Department of Computer Engineering, National Institute of Technology, Kurukshetra,
Haryana, India

Ritu.59@gmail.com

ABSTRACT

Fault tolerance is an important property for large scale computational grid systems, where geographically distributed nodes co-operate to execute a task. In order to achieve high level of reliability and availability, the grid infrastructure should be a foolproof fault tolerant. Since the failure of resources affects job execution fatally, fault tolerance service is essential to satisfy QOS requirement in grid computing. Commonly utilized techniques for providing fault tolerance are job checkpointing and replication. Both techniques mitigate the amount of work lost due to changing system availability but can introduce significant runtime overhead. The latter largely depends on the length of checkpointing interval and the chosen number of replicas, respectively. In case of complex scientific workflows where tasks can execute in well defined order reliability is another biggest challenge because of the unreliable nature of the grid resources.

KEYWORDS

Grid Computing, Fault Tolerance, Workflow Grid

1. INTRODUCTION

Computational grid [1] consists of large sets of diverse, geographically distributed resources that are grouped into virtual computers for executing specific applications. As the number of grid system components increases, the probability of failures in the grid computing environment becomes higher than that in a traditional parallel computing scenario [8, 9, 10]. Compute intensive grid applications often require much longer execution time in order to solve a single problem. Thus, the huge computing potential of grids, usually, remains unexploited due to their susceptibility to failures like, process failures, machine crashes, and network failures etc. This may lead to job failures, violating timing deadlines and service level agreements, denials of service, degraded user expected quality of service. Thus fault management is a very important and challenging for grid application developers. It has been observed that *interaction*, *timing*, and *omission* faults are more prevalent in grid.

Fault tolerance is the ability of a system to perform its function correctly even in the presence of faults. The fault tolerance makes the system more dependable. A complementary but separate approach to increase dependability is fault prevention. This consists of techniques, such as inspection, whose intent is to eliminate the circumstances by which faults arise. A failure occurs when an actual running system deviates from this specified behavior. The cause of a failure is called an error. An error represents an invalid system state that does not comply the system specification. The error itself is the result of a defect in the system or fault. In other words, a fault is the root cause of a failure. However, a fault may not necessarily result in an error; nevertheless, the same fault may result in multiple errors. Similarly, a single error may lead to multiple failures.

The level of fault tolerance is reflected by quantifying the system dependability. Dependability means that our system can be trusted to deliver the service(s) for which it has been designed. It can be measured by two of the metrics like reliability and availability. *Reliability* characterizes the ability of a system to perform, on demand, its service correctly. Availability means that the system is up to perform this service when it is asked to do so. Technically, reliability is defined as the probability that a system will perform correctly up to a given point in time. Closely related to reliability are the mean time to failure(MTTF) and mean time between failures (MTBF).The first is the average time the system operates until a failure occurs, whereas the second is the average time between two consecutive failures.

$MTBF = MTTF + MTTR$, where MTTR is the mean time to repair

Availability is defined as the probability that a system is operational at desired time. For a given system, this characteristic is strongly dependent on the time it takes to restore it to service after some failure.

$Availability = (MTTF) / (MTTR + MTTF)$

The paper is organized as follows. Section II briefly describes the different approaches used for handling grid. Section III presents the fault tolerant techniques like replication and check pointing in detail. Section IV provides how fault management is done in dependency grid (workflow grid). Section V presents the conclusion and Section VI presents some of the open issues related to grid fault tolerance which can be explored further.

2. GRID FAULT MANAGEMENT

Various approaches are used for tolerating faults in grid, so grid fault management can be classified as

2.1. Pro-active vs. Post-active management

In literature, the work on grid fault tolerance can be divided into pro-active and post-active mechanisms. In pro-active mechanisms, the failure consideration for the grid is made before the scheduling of a job, and dispatched with hopes that the job does not fail. Whereas, post-active mechanisms handles the job failures after it has occurred. However, in the dynamic systems only post-active mechanism is relevant [13].

2.2. Push Model vs. Pull Model

In order to detect occurrence of fault in any grid resource two approaches can be used: the push or the pull model. In the *push model*, grid components periodically send heartbeat messages to a failure detector, announcing that they are alive. In the absence of any such message from any grid component, the fault detector recognizes that failure has occurred at that grid component. It then implements appropriate measures dictated by the predefined fault tolerance mechanism. In contrast, in the *pull model* the failure detector sends live-ness requests (“*Are you alive?*” messages) periodically to grid components. [16].

2.3. Grid Middleware Based

Like any middleware, a grid middleware is also responsible to hide, from the application developer, the technical details related to different syntax and access methods and to provide a consistent and homogeneous access to resources managed locally. Hwang et al. [18] presented a failure detection service (FDS) and a flexible failure handling framework (Grid-WFS) as a fault tolerance mechanism on the grid. The FDS enables the detection of both task crashes and user-defined exceptions.

2.4. Agent based

Here autonomous, light-weight, intelligent agents monitor with individual faults. Agents maintain log of various informations related to hardware conditions, memory utilization, resource constraints, network status and component failure. Based on this information and critical states, agent can enhance the reliability and efficiency of grid services e.g., Mohamad et al. [19] use agents to inject proactive fault tolerance in grids.

2.5. Application level

The performance of system level fault tolerance mechanisms is restricted due to the large process state, low I/O bandwidth, and the high frequency of failures. Either an application would spend more time in taking checkpoints or it does not get sufficient time to save its core to disk before the next failure occurs. Therefore, low overhead application level fault tolerance schemes are more suitable alternative in long-lived computational tasks. However, most application level fault tolerance schemes proposed in literature are non-adaptive due to fast changing computational scenario. However, in order to achieve high reliability and survivability, the fault tolerance schemes in such applications need to be adaptable to dynamic system environments. The CPPC (Controller/Precompiler for Portable Checkpointing) framework [20] implements checkpointing module into the application code. The CPPC-G service is responsible for (a) the submission and monitoring of the application execution (b) the management of checkpoint files generated by CPPC-enabled applications, and (c) the detection and automatic restart of failed executions.

2.6 Fault tolerance at job site

The failure at job site has cascading effect on the grid performance. They reduce the resource availability, which tend to make the clusters unusable that result in the loss of user submitted jobs. It eventually slows down the overall speed of computation. Hence, in order to ensure the high system availability, the job site failure handling is inevitable. A cluster head is responsible to coordinate the computation related activities and to provide other necessary services, such as job scheduling. If job sites are made up of clusters, then the failure of the cluster head causes the services to be unavailable till the cluster head recovers or replaced by some back up head. HA-OSCAR [21] uses redundancy and self healing techniques to address the problem of single point failure. The transition from failed head to back up head is uninterrupted. Hence, it provides an excellent solution for stateless services. While some approaches [23, 24, 25] are self healing, provides job level fault resilience in order to guarantee the high availability of the site.

3. FAULT RESILIENCE

Checkpoint-recovery and job replication are two primarily used fault tolerance techniques in cases of a system outage. The first approach depends on the system's MTTR while the latter depends on the availability of alternative sites to run replicas.

3.1 Checkpointing

The checkpointing is one of the most popular technique to provide fault-tolerance on unreliable systems. It is a record of the snapshot of the entire system state in order to restart the application after the occurrence of some failure. The checkpoint can be stored on temporary as well as stable storage [26]. However, the efficiency of the mechanism is strongly dependent on the length of the checkpointing interval. Frequent checkpointing may enhance the overhead, while lazy checkpointing may lead to loss of significant computation. Hence, the decision about the

size of the checkpointing interval and the checkpointing technique is a complicated task and should be based upon the knowledge about the application as well as the system. Therefore, various types of checkpointing optimization have been considered by the researchers, e.g., (i) Full checkpointing or Incremental checkpointing (ii) Unconditional periodic checkpointing or Optimal (Dynamic) checkpointing (iii) Synchronous (Coordinated) or asynchronous (Uncoordinated) checkpointing, and (iv) Kernel, Application or User level checkpointing.

3.1.1 Full Checkpoint or Incremental checkpoint

A full checkpoint is a traditional checkpoint mechanism which occasionally saves the total state of the application to a local storage. However, the time consumed in taking checkpoint and the storage required to save it is very large.

Incremental checkpoint mechanism was introduced to reduce the checkpoint overhead by saving the pages that have been changed instead of saving the whole process state [27, 30, 31]. In the incremental checkpoint scheme, the first checkpoint is typically a full checkpoint. After that, only modified pages are checkpointed at some predefined interval. When large numbers of pages get modified another full checkpoint is taken. In order to recover the application, we will load a saved state from the last full checkpoint and load the changed pages from each incremental checkpoint following the last full checkpoint. This results in more expensive recovery cost than the recovery cost of the full checkpoint mechanism.

3.1.2 Uncoordinated or Coordinated checkpointing

In uncoordinated checkpointing each process takes its checkpoint independently of the other processes though it may lead to domino effect (processes may be forced to rollback up to the execution beginning). Since there is a chance for losing the whole computation, these protocols are not popular in practice.

Coordinated checkpoint protocols produce consistent checkpoints; hence, the recovery process is simple to implement. Communication Induced Checkpointing (CIC) tries to take advantage of uncoordinated and coordinated checkpoint techniques. Based on the uncoordinated approach, it piggy backs causality dependencies in all messages and detects risk of inconsistent state. When such a risk is detected, some processes are forced to checkpoint. A detailed survey of checkpointing protocols may be found in [32].

3.1.3 Kernel or Low level checkpointing

Here checkpointing procedures are included in the kernel, checkpointing is transparent to the user and generally no changes are required to the programs to make them checkpointable. When the system restart after failure, the kernel is responsible for managing the recovery operation. To date, there have been a few low-level checkpointing packages [34]. Each checkpointing package offers a different functionality and interface. Because of technical issues the checkpointing packages impose some limitations on applications that are to be checkpointed. So the integration of low-level checkpointing packages with the Grids is a difficult task. AltixC/R [36] is kernel-level checkpointing package. The required kernel-level code is provided in a form of a dynamically loaded kernel module so it is easy to use and install. The package is able to checkpoint multi-process programs.

3.1.4 User level Checkpointing

In this approach, a user level library is provided to do the checkpointing. To checkpoint, application programs are linked to this library. This approach generally requires no changes in the application code; however explicit linking is required with user level library, which is also responsible for recovery from failure.

3.1.5 Application level checkpointing

Here, the application is responsible for carrying out all the checkpointing functions. Code for checkpointing and recovery from failure is written into the application. It is expensive to implement but provide more control over the checkpointing process.

3.2 Replication

It is a technique based on an inherent assumption that any single resource is much susceptible to failure as compared to simultaneous failure of multiple resources. Unlike checkpointing, the replication avoids task re-computation by executing several copies of the same task on more than one compute stations. The job replication and determination of the optimal number of replicas involves many technical considerations. The task replication in grids has been studied in [38]. A number of approaches have been used to implement replication in grid computing environment.

3.2.1 Static vs. Dynamic replication

The static replication [39] means that, when some replica fails, it is not replaced by a new one. The number of replicas of the original task is decided before execution. While in case of dynamic replication, new replicas can be generated during run time. Gallop [40] used an initial set of replicas based on user preferences which we call static replicas (e.g. the user wants x replicas). The user may also indicate that they do not want the number of active replicas falling below y replicas. This may require that new replicas are started if enough sites fail and the number of active replicas falls below y . We call this capability dynamic replication.

3.2.2 Active vs. passive replication

In the former, the state of replicas is kept closely synchronized; replicas service the same requests in parallel and undergo the same state transitions. This algorithm is referred to as the active replication [41]. In the latter, a primary replica services requests on behalf of clients. Other replicas are kept as standby and can take over in the case of a primary failure [42]. This is sometimes referred to as passive replication. Further, two approaches of passive replication, using the concept of overloading, have been used in the literature; Primary Backup vs. Backup Backup overloading.

3.2.3 Primary Backup vs. Backup Backup overloading Replication

Overloading techniques are used to deal with timing faults and to improve the schedulability. In PB-overloading primary of a task is scheduled onto the same or overlapping time slot with the backup of another task on a processor. While in BB-overloading backups of multiple tasks are scheduled onto the same or overlapping time slot on a processor [43]. Since PB-overloading can assign an earlier start time than that of the BB-overloading, thus increasing the schedulability In [44], R. Al-Omari et al. concluded that the PB-overloading is able to achieve better performance than BB-overloading, and BB-overloading algorithm is better than no-overloading.[45]. In short, hybrid overloading is a new technique which combines the advantages of both PB and BB overloading. All three overloading strategies are compared through a stochastic analysis, as well as by simulating them under diverse system conditions.

4. TASK DEPENDENCY OF AN APPLICATION

When the relations among the tasks in the grid application are considered, a common dichotomy used is dependency vs. independency. All previously mentioned techniques have been used in the independent task scenario. Usually dependency means there are precedence orders existing in the tasks, that is, a task cannot start, or sometimes can't progress, until its predecessors are done. Dependency has crucial impact on the fault tolerance.

4.1 Fault tolerance in dependent task Grid (or Workflow Grid)

Grid workflow is defined as the orchestration of a set of atomic tasks processed at distributed resources in a well-defined order to accomplish a large and sophisticated goal. Currently, Directed Acyclic Graph (DAG) has been extensively used in scientific computational workflow modelling. In a Grid environment, workflow execution failure can occur for various reasons: the variation in the execution environment configuration, non-availability of required services or software components, overloaded resource conditions, system running out of memory, and faults in computational and network fabric components. Grid workflow management systems should be able to identify and handle failures and support reliable execution in the presence of concurrency and failures [47, 48]. Hwang et al. [51] divided workflow failure handling techniques into two different levels, namely task-level and workflow-level. Task-level techniques handle the execution failure of tasks like task independent scenario and the techniques used are also similar as discussed above, while workflow-level techniques may alter the sequence of execution in order to address the failures. Hwang and Kesselman proposed three different techniques on the basis of assumption that there is more than one implementation possible for a certain computation with different execution characteristics.

- (i) The alternate task technique executes another implementation of a certain task if the previous one failed.
- (ii) The redundancy technique executes multiple alternative tasks simultaneously.
- (iii) The user-defined exception handling allows the users to specify a special treatment for a certain failure of a task in workflow.
- (iv) The rescue workflow technique developed in Condor DAGMan system [52] ignores the failed tasks and continues to execute the remainder of the workflow until no more forward progress can be made. It uses a rescue workflow description called rescue DAG in order to indicate failed nodes with statistical information that can be used for further processing.

5. CONCLUSION

In the light of above survey, fault tolerance plays an important role in order to achieve availability and reliability of a grid system. Replication, Check pointing and job migration are the major techniques used in any fault-tolerant grid management system. Replication provides better reliability and improved response time. The ability to checkpoint a running application and restart it later can provide many useful benefits like fault recovery, advanced resources sharing, dynamic load balancing and improved service availability. In case of dependent task grid (workflow grid) fault tolerance can be handled at two levels i.e. task level and workflow level.

6. OPEN ISSUES

- While using job migration approach for fault tolerance, the jobs are allowed to migrate within area of their designated cluster. Although, this approach restricts overheads, it leads to under utilization of the grid resources, which might be available in some other cluster. New job migration protocols are yet to come that allows intra-cluster migration, which may enhance the reliability by sharing of checkpoint images between distant clusters.
- Many works on fault tolerance use adaptive as well as load balanced techniques. However, they try to plan the flow of execution before it starts. In real life environments, when there is sudden increase of load on the system, it tends to increase the number of failures, which eventually increases the system downtime. On demand fault tolerant techniques are required which can handle such burst in load and do not cause overhead in absence of burst.

- Since grids are highly dynamic in nature, so must handle failure in the resources and check how changes in the topology and computational capability of the grid resources affect the efficiency in terms of deadline of the tasks.
- In case of fault treatment, using replication and other techniques, it is interesting to develop a viable economic model that could provide an execution environment, which guarantees certain minimum cost and time even in the presence of failures, unlike present fault tolerant approaches where, usually, cost increases exponentially.
- A model could be designed and developed that would estimate the expected profit and decide which strategy would be ideal to follow in terms of performance and profit optimization.
- Since grid is difficult, more complex to implement and manage, so there could be the differences in their performance under diverse experimental conditions. We should modify the algorithm so that such differences are not noticeable.
- Most of the approaches of fault tolerance are based on the prediction of failure probability of the resources in certain time interval. It is hard to achieve the resource failure prediction even with years of historic trace data. Hence, efficient techniques are required which do not base their decisions on the specific failure model and do not rely on the failure prediction accuracy.
- The existing reliability and fault tolerance model may be augmented by considering scalability and security into consideration.
- We can go for development of workflow applications models that can balance properly between reliability performance and resource usage. We can extend workflow grid fault tolerance in more complex scenario such as those in which background load is present on the resources along with the multiple fault tolerant instances on the same grid.

REFERENCES

- [1] I. Foster, C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers, Los Altos, CA, 1998.
- [2] F. Berman, G. Fox, and T. Hey, "Grid Computing: Making the Global Infrastructure a Reality. Chichester", John Wiley & Sons, 2003.
- [3] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid", In Proceedings of 4th International Conference on High Performance Computing in Asia-Pacific Region, Beijing, China, 2000.
- [4] K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", In Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing, 2001 .
- [5] I. Foster and C. Kesselman, S.T., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", Intl. J. Supercomputer Applications, 2001.
- [6] E. Beguelin, Seligman and P. Stephan, "Application Level Fault Tolerance in Heterogeneous Networks of Workstations", Journal of Parallel and Distributed Computing on Workstation Clusters and Networked-based Computing, Vol. 43(2), 1997, pp. 147-155.
- [7] K Limaye, B. Leangsuksun, Z. Greenwood, S. L. Scott, C. Engelmann, R. Libby and K. Chanchio, "Job-Site Level Fault Tolerance for Cluster and Grid environments" In Proceedings of the IEEE international conference on cluster computing, 2005, pp. 1-9.
- [8] P. Stelling, I. Foster, C. Kesselman, C. Lee, G. von Laszewski, "A fault detection service for wide area distributed computations", In: Proceedings of 7th IEEE Symposium on High Performance Distributed Computing, 1998.

- [9] S. Vadhiyar, J. Dongarra, "A performance oriented migration framework for the grid", In: Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003.
- [10] F. Vraalsen, R. Ayt, C. Mendes, D. Reed, "Performance contracts: predicting and monitoring grid application behavior", in: Proceedings of the 2nd International Workshop on Grid Computing, 2001.
- [11] M. C. Elder, "Fault Tolerance in Critical Information Systems", Ph.D. thesis, University of Virginia, 2001.
- [12] I. Foster, "What is the Grid? A Three Point Checklist", GRIDToday, 2002.
- [13] R. Medeiros, W. Cirne, F. Brasileiro, J. Sauve, "Faults in grids: why are they so bad and what can be done about it?" In proceedings of the 4th international workshop, November 2003, pp 18–24.
- [14] R. Rabbat, T. McNeal, and T. Burke, "A High-Availability Clustering Architecture with Data Integrity Guarantees"; In Proceedings of the IEEE International Conference on Cluster Computing, 2001. pp. 178-182.
- [15] S. Ghosh, R. Melhem, D. Mosse, "Fault-tolerance through scheduling of a periodic tasks in hard real-time multiprocessor systems", IEEE Transactions on Parallel and Distributed Systems, Vol.8 (3), 1997, pp.272-284.
- [16] Y. Li, Z. Lan, "Exploit failure prediction for adaptive fault-tolerance in cluster". In: Proceedings of the sixth IEEE International symposium on cluster computing and the grid, Vol 1, May 2006, pp. 531-538.
- [17] H. Lee , K. Chung , S. Chin , J. Lee , D. Lee , S. Park , H. Yu, "A resource management and fault tolerance services in grid computing.", J Parallel Distrib Computing, Vol. 65(11), 2005, pp. 1305–1317
- [18] S. Hwang and C. Kesselman, "A Generic Failure Detection Service for the Grid", Technical Report ISI-TR-568, USC Information Sciences Institute, 2003.
- [19] Mohammad Tanvir Huda, Heinz W. Schmidt, Ian D. Peake, "An Agent Oriented Proactive Fault-Tolerant Framework for Grid Computing", In Proceedings of the First International Conference on e-Science and Grid Computing, 2005, pp.304-311.
- [20] G. Rodriguez, M. J. Martin, P. Gonzalez, and J. Tourino. "Controller/Precompiler for Portable Checkpointing", IEICE Transactions on Information and Systems, Feb. 2006, pp.408–417.
- [21] C. Leangsuksun, Tong Liu, Tirumala Rao, Stephen L. Scott, and Richard Libby , "A Failure Predictive and Policy-Based High Availability Strategy for Linux High Performance Computing Cluster", In Proceedings of the 5th LCI International Conference on Linux Clusters, 2004.
- [22] L. He, S.A. Jarvis, D.P. Spooner, H. Jiang, D.N. Dillenberger, G.R. Nudd, "Reliability driven task scheduling for heterogeneous systems", In: Proceedings of the Lasted International Conference on Parallel and Distributed Computing and Systems, 2003, pp. 465-470.
- [23] J. B. Weissman and D. Womack, "Fault tolerant scheduling in distributed networks", Technical Report CS-96-10, Department of Computer Science, University of Virginia, Sep.1996.
- [24] J. H. Abawajy, "Fault-tolerant scheduling policy for grid computing systems", In Proceedings of the International Parallel and Distributed Processing Symposium, IEEE Computer Society, Los Alamitos, United States, 2004, pp.3289–3295.
- [25] Paul Townend, Jie Xu, " Fault Tolerance within Grid environment", In Proceedings of the AHM2003, <http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/063.pdf>, 2003, pp. 272-275.
- [26] Oliner, A.J., Sahoo, R.K., Moreira, J.E., Gupta, M.: "Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems", In Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Washington, 2005.

- [27] S. Agarwal, R. Garg, M. Gupta, and J. Moreira, "Adaptive Incremental Checkpointing for massively Parallel Systems," In Proceedings of the 18th Ann. International Conf. Supercomputing , Nov. 2004.
- [28] Y. Liu, C. B. Leangsuksun, "Reliability-aware Checkpoint /Restart Scheme: A Performability Trade-off", In Proceedings of the International conference of IEEE Cluster Computing, Boston, MA, September 2005.
- [29] A. Girault, C. Lavarenne, M. Sighireanu, Y. Sorel, "Fault-tolerant static scheduling for real-time embedded systems", In: Proceedings of the Int. Conf. on Computing Systems, April 2001.
- [30] A. C. Palaniswamy, and P. A. Wilsey, "An analytical comparison of periodic checkpointing and incremental state saving", In Proc. of the Seventh Workshop on Parallel and Distributed Simulation, California, 1993, pp. 127-134.
- [31] J.S. Plank, J. Xu, and R.H. Netzer, "Compressed differences: an algorithm for fast incremental checkpointing", Technical Report CS-95-302, University of Tennessee at Knoxville, 1995.
- [32] L. Alvisi, E. Elnozahy, S. Rao, S. A. Husain, and A. D. Mel. "An analysis of communication induced checkpointing", In Proceedinds of the 29th Symposium on Fault-Tolerant Computing, IEEE CS Press, June 1999.
- [33] F. Baude, D. Caromel, C. Delbe and Ludovic Henrio, "A Hybrid Message Logging-CIC Protocol for Constrained Checkpointability", In Proceedings of the EuroPar , Lisbon, Portugal, September 2005.
- [34] Eric Roman, "A survey of Checkpoint/Restart Implementations", Lawrence Berkley National Laboratory, CA, 2002.
- [35] J. Daly. "A model for predicting the optimum checkpoint interval for restart dumps", In Proceedings of the ICCS2003, Lecture Notes in Computer Science 2660 ,Vol 4, 2003, pp.:3–12.
- [36] Gracjan Jankowski, Radoslaw Januszewski, Rafal Mikolajczak, "Checkpoint/Restart mechanism for multiprocess applications", implemented under SGIGrid Project, CGW2004. <http://checkpointing.psnc.pl/>
- [37] Kshitij Limaye, Box Leangsuksun, "HAOSCAR: "Grid enabled High availability framework", In Proceedings of the 13th Annual Mardi Gras conference, Frontiers of Grid Applications and Technologies, 2005.
- [38] M. Chtepen, F. H. A. Claeys, B. Dhoedt, F. De Turck, P. Demeester, P.A. Vanrolleghem, "Evaluation of Replication and Rescheduling Heuristics for Grid Systems with Varying Availability". In Proceedings of the Parallel and Distributed Computing and Systems, Dallas, 2006.
- [39] A. Nguyen-Tuong, "Integrating fault-tolerance techniques in Grid applications", Ph.D. Dissertation, University of Virginia, August 2000.
- [40] J.B. Weissman, "Gallop: The Benefits of Wide-Area Computing for Parallel Processing," Journal of Parallel and Distributed Computing, Vol. 54(2), November 1998.
- [41] Fred B. Schneider, "Replication Management using the State-Machine Approach", ACM Computing Surveys, Vol.22, 1990.
- [42] Navin Budhiraja, Keith Marzullo, Fred B. Schenider, and Sam Toueg, "The Primary-Backup Approach", In Distributed Systems. Addison-Wesley, Second Edition 1993.
- [43] S. Ghosh, R. Melhem, D. Mosse, "Fault-tolerance through scheduling of a periodic tasks in hard real-time multiprocessor systems", IEEE Trans. Parallel distributed Systems, Vol.8 (3), March 1997, pp.272–284.
- [44] R. AlOmari, A.K. Somani, G. Manimaran, "Efficient overloading techniques for primary-backup scheduling in real-time system," J. Parallel and Distributed Computing, Vol. 64, 2004, pp. 629-648.

- [45] W. Sun, C. Yu, X. Défago and Y. Inoguchi, "Hybrid Overloading and Stochastic Analysis for Redundant Scheduling in Real-time Multiprocessor Systems", In Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems, 2007, pp.265-274.
- [46] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments", Journal of Grid Computing, Vol. 1(1), 2003, pp. 25–39.
- [47] Gosia Wrzesinska, Rob V. van Nieuwport, Jason Maassen, Thilo Kielmann, and Henri E. Bal, "Fault-tolerance scheduling of fine grained tasks in Grid environment", International Journal of High Performance Applications .Vol 20(1), 2006, pp.103-114.
- [48] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing, Journal of Grid Computing, Springer Science+Business Media B.V., New York, USA, Vol 3(3-4),Sept. 2005, pp.171-200.
- [49] P. Kacsuk, G. Sipos, "Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal", Journal of Grid Computing, Feb 2006, pp. 1-18.
- [50] D. Hollingsworth, "Workflow Management Coalition: The Workflow Reference Model", WfMC-TC00-1003, 1994.
- [51] S. Hwang and C. Kesselman. "Grid Workflow: A Flexible Failure Handling Framework for the grid", In Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, Seattle, Washington, USA., IEEE CS Press, Los Alamitos, CA, USA, June, 2003.
- [52] DAGMan Application. http://www.cs.wisc.edu/condor/manual/v6.4/2_11
DAGman_Applicaitons.html [December 2004]

Authors

Ritu Garg received B.Tech degree in Computer Science from Punjab Technical University, Jalandhar, India in 2001. She received M.Tech in the area of Computer Science from Kurukshetra University, Kurukshetra in 2006. Currently, she is pursuing PhD in the area of Resourse Management in Grid Computing from National Institute of Technology, Kurukshetra, India. Her research interests include Grid Computing, Scheduling and Fault Tolerance.

Awadhesh Kumar Singh received B. E. degree in Computer Science & Engineering from Gorakhpur University, Gorakhpur, India in 1988. He received M.E. and PhD (Engg) in the same area from Jadavpur University, Kolkata, India. Currently, he is Associate Professor in the Department of Computer Engineering, National Institute of Technology, Kurukshetra, India. His present research interest is mobile distributed computing systems.