THE MEDIAN RESOURCE FAILURE CHECKPOINTING

Suleman Khan, Khizar Hayat, Sajjad A. Madani
COMSATS Institute of Information Technology (CIIT),
Abbottabad 22060, Pakistan.
Email: sulemankhan1984@yahoo.com, khizarhayat@ciit.net.pk, madani@ciit.net.pk

Samee U. Khan
Department of Electrical and Computer Engineering, North Dakota State University,
Fargo, ND 58108-6050, USA.
Email: samee.khan@ndsu.edu

Joanna Kolodziej
Department of Mathematics and Computer Science, University of Bielsko-Biala,
PL-43300 Bielsko-Biala, Poland.
Email: jkolodziej@ath.bielsko.pl

KEYWORDS

Fault tolerance, Checkpointing, Distributed systems

ABSTRACT

In grid computing, the realization of an enviable fault tolerance ability is linked with the proper utilization of resources and scheduling of jobs. The literature offers two solutions to these two challenging tasks, *viz.* checkpointing and replication. A checkpointing strategy is being proposed that uses the median of failure intervals of the resources in deciding the checkpoint intervals for the given jobs. The strategy shows improved system throughput, job losses and job execution times while eliminating unnecessary checkpoints.

1 INTRODUCTION

It is a usual human tendency to pay little attention to the aspect of fault tolerance, while designing a system. Take, for example, your electronic wrist watch, if it misreports the time costing you some important assignment, the focus of your fury would be the weak battery rather than the watch. The blame is mainly on the watch maker who, for the sack of economy, has ignored the fault tolerance facet and has not put much effort to incorporate any functionality to alert a user in case of low battery. On the other hand, an ordinary car possesses a fault tolerant mechanism against low battery, for the dividends it has for its manufacturer. In a nutshell, a fault tolerance mechanism is generally considered valuable for a system if it adds to the utility of the corresponding system.

When it comes to the distributed grid environments, the concept of fault tolerance becomes rather more important. Due to the heterogeneous nature of the underlying infrastructure, the resources are geographically dispersed in these systems. These resources may be executed under different administrative domains, each of which may behave and deal, even a similar nature of jobs, differently. Handling of such a dynamic and huge environment is a challenging job and requires innovative efforts to minimize the fault incidents. System faults can be variously classified based on their nature, length and cause of occurrence. The nature of a fault in the system can be attributed to the hardware failure, operating system failure, network failure, or IO failure. A system going down due to the aforementioned failures may lead to catastrophic results. The most common causes of occurrence of system fault are incompatibility of various devices, improper software, and external intrusions (Johnson, 1996).

Fault tolerance, in a grid environment, is dependent on the apt utilization of resources as well as on the balanced scheduling of jobs. The grid computing literature offers two solutions to implement proper resource utilization and scheduling, namely the checkpointing (Wong and Franklin, 1996; Cao and Singhal, 2003; Deng and Park, 1994) and replication (Narasimhan et al., 2000; Saito and Levy, 2000; Ratner et al., 1999). Both these approaches have their disadvantages when used in the static mode. With the former, when checkpointing requests are generated, it stops the job and saves its previous state on a stable resource. This may consume a fair amount of time in generating, storing and recovering the checkpoints back. On the other hand, although replication produces its replicas on free available computational resources giving more chance for a job to be executed; the execution of more replicas in a resource-poor distributed environment may lead to low throughput and high job execution times. In this work we focus on the checkpointing aspect and propose a strategy based on our median resource failure checkpointing (MRFCP) algorithm.

The paper is organized as follows: Section 2 explains the related work followed by a discussion on the proposed method in Section 3. The simulation results are presented in Section 4 whereas the conclusion is given in Section 5.

Proceedings 26th European Conference on Modelling and Simulation ©ECMS Klaus G. Troitzsch, Michael Möhring, Ulf Lotzmann (Editors)
ISBN: 978-0-9564944-4-3 / ISBN: 978-0-9564944-5-0 (CD)

2 RELATED WORK

Checkpointing is used to save the executed portion of the jobs running on a resource in case of anticipated resource failure (Pruyne and Livny, 1996). When a checkpoint request is generated, an executable portion of the job states are stored on stable resources (Bouabache et al., 2008). After a resource failure, a job is migrated to the some other available resource(s) for its further execution; it rebuilds the previous states of the job by recalling from the stable resources. The main advantage of using checkpointing is that a job does not start its execution from the start whenever a resource fails. On the other hand, extensive checkpoints requests may lead to overheads in terms of latency, job execution time and system management (Plank et al., 1995).

The scheme, given in (Silva and Silva, 1998), does not save a checkpoint on a disk while using the main memory of its neighboring processors. When a checkpoint request is generated, it saves the states of the jobs in its main memory as well as in its neighbor's. All the resources are arranged in a virtual ring. Each resource has only one connection to it neighboring resource for keeping a record of checkpoints. Each resource should keep two slots of space for the checkpoints; one for its own local states and other for its preceding neighbor resource. The technique is simple and the nature of its failures is not that serious. In addition, it is memory intensive, resulting in high overhead in the shape of system complexity for larger grid environments.

The coordinated checkpointing strategy of (Tamir and Squin, 1984) introduces the concept of a coordinator which stops the processing of a job while taking the image of the job on resource. It also broadcast the message to all the resources to stop their processes and take the checkpoints. Each resource, in return, sends an acknowledgment message to the coordinator telling that the checkpoint process has been successfully completed. Thereupon a success message is broadcast, by the coordinator, to all the resources who then discard the existing checkpoint image and update their individual tables. Due to the involvement latency overheads in the process, de-blocking of the checkpointing is preferred on part of many resources, in practice, thus violating the agreed policy (Elnozahy and Zwaenepoel, 1992). A remedy is therefore inevitably needed to avoid the latency overhead. In (Koo and Toueg, 1987), the overhead is minimized to some extent by adopting a two phase protocol which reduces the coordinated checkpointing. In the first stage, the checkpoint initiator sends a message to those jobs to which it communicated during the last checkpointing process. These jobs, in turn, send the message to all those jobs with which they communicated in their last specified checkpoint stage. This process further continues until all jobs which were involved in the last checkpoint receive a message. In the second stage, all the processes which were identified in the first stage make checkpoints. This results in consistent checkpoints with lesser overhead.

An incremental checkpointing concept has been introduced in (Agarwal et al., 2004), in which the data is stored in a block of memory which has been modified since the last checkpoint wave. A protocol is used which directly distributes the checkpoint images in memory of its computer peers, as in the FT-MPI[1] project or Charm++[2] project. An advantage of this approach is that it does not stop the other processes which were not part of the last checkpointing process. The strategy creates consistency of checkpoint images among those jobs which were part of the last checkpointing process. This reduces lot of overhead and economizes a job's execution time. Bouguerra et al. (2010) propose their "coordinated Checkpoint/Restart mechanism" based on three factors, namely the process failure distribution, the cost to save a global consistent state of processes and the number of computational resources. Relying on a reliability analysis, the authors employ their mechanism to ascertain the optimal interval between checkpoint times while minimizing the average completion time. The authors claim about 20% improvement in the checkpoint rate through the adoption of their proposed model. Another checkpointing scheme is outlined in Bouguerra et al. (2011) that mainly deals in batch jobs under the constraint that failures obey a general probability distribution.

3 THE PROPOSED METHOD

The grid environment scenario considered to elaborate the proposed strategy is shown in Fig. 1. It consists of four distributed heterogeneous sites with 32 computational resources each. The given sites are connected through a wide area network and computational resources within site are connected through a local area network. The users, through a user interface (UI), submit their jobs to the grid transparently. A resource broker is responsible to assign the user jobs to the available and suitable resources. Table 1 describes the symbols used in the explanation of the proposed strategy. The median

Table 1: List of symbols

Symbol	Description
RT_r^J	Remaining time of the job J on resource r
MRF_r	Median failure interval of r
ET_r^J	Execution time of J on r
I	Interval
CI	Checkpoint interval
CI_r^J	Checkpoint interval of J on r
$CI_r^{J^{new}}$	New selected checkpoint interval for J on r
$CI_r^{J^{old}}$	Old checkpoint interval for J on r
CRD	Checkpoint run-time delay
∂	Fraction of J w.r.t. total

resource failure checkpointing (MRFCP) algorithm is being proposed for the modification of the initially specified

[1] http://icl.cs.utk.edu/ftmpi/

[2] http://charm.cs.uiuc.edu

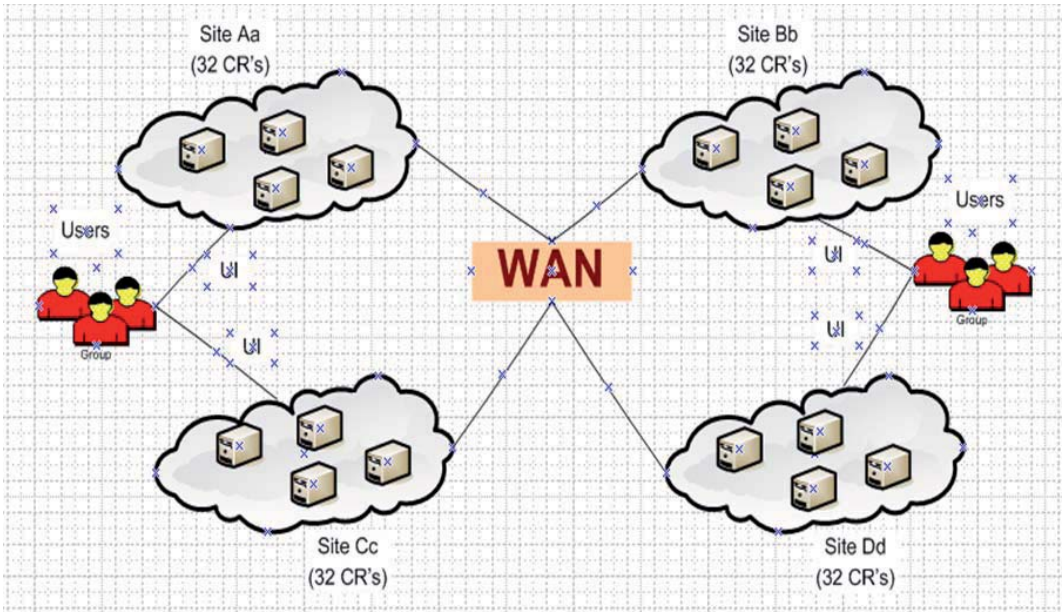

Figure 1: A Grid Computing Scenario for the proposed strategy

Input: Checkpoint request for a resource r, running several jobs
Output: Checkpoint intervals ($CI_r^{J^{new}}$)

```
1 begin
2   if RT_r^J < MRF_r AND CI_r^J < ∂ * ET_r^J then
3       /* ∂ < 1                                          */
4       set CI_r^{J^new} ← CI_r^{J^old} + CI
5   else
6       set CI_r^{J^new} ← CI_r^{J^old} − CI
7   end
8 end
```

Algorithm 1: The MRFCP algorithm

static checkpoint intervals. The algorithm is based on the comparison of the remaining time (RT) of the job on a resource and its median resource failure (MRF) interval whereupon it decides on the running time. MRF captures the whole failure event of the resource and whenever the system request for checkpoints, it takes all the failure events time of the resource and take its median value. This median value is then compared with the remaining time of the job to decide the increase/decrease of the checkpoint intervals. A pseudocode description of the proposed MRFCP strategy is given in Algorithm 1.

An instance for the MRFCP algorithm of one job resource execution is depicted in Fig. 2. When RT_r^J is less than MRF_r, it increases the checkpoint interval while decreasing the frequency of the checkpoint. This indicates that resource r is stable enough to execute a job or job J has almost finished its entire execution. The second condition shows that the checkpoint interval should not exceed the job length. This may be important for small jobs. If this condition is not fulfilled the algorithm decreases the checkpoint interval while increasing the frequency of checkpoints.

4 SIMULATION RESULTS

Performance metrics

For the comparison of MRFCP with the existing checkpointing strategies, three standard metrics were employed, namely the average job execution time, Successful job execution and average number of checkpoints.

1. *Average job execution time:* is the total execution time of successful jobs divided by total number of successful jobs. It is also called average turnaround time of the system.

2. *Successful job execution:* it is calculated by subtracting number of failed jobs from total number of submitted jobs in the system. This indicates the system throughput that how much system has successfully executed the jobs.

3. *Average number of checkpoints:* this metric is calculated by dividing total number of successful jobs on total number of checkpoints of successful jobs. Checkpoint requires time to take place, store, and to retrieve.

Simulation setup

For the simulations, the GridSim 5.2 simulator (Buyya and Murshed, 2002) had been employed. Besides being an effective tool to model different heterogeneous resources, schedulers, users and its applications, the GridSim simulator has the facility to simulate scheduler for single and multiple administrative domains in distributed environments. The resources can be scheduled in two modes, namely the space-share mode and the time-share mode. In our simulation we have used space-share mode for resources. Moreover, the resource strength can be

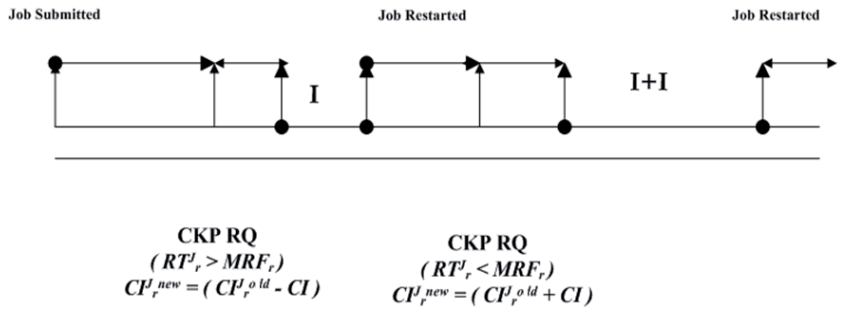

Figure 2: MRFCP running on single job resource

measured in million instructions per second (MIPS). In addition, the resources can not only be mapped into different time zones but weekends and holidays can also be mapped in the GridSim simulation environment.

Table 2: The simulation setup

Type	Parameter	Value
Network	LAN links	
	- Bandwidth	56 $Mbps$
	- Latency	1 ms
	WAN links	
	- Bandwidth	100 $Mbps$
	- Latency	$3 - 10$ ms
	Number of cluster zones	4
	Nodes per zone	32
	Propagation delay for IS	10 min
Host	Speed	10 $MIPS$
	Parallel execution	2 Jobs
	Failure model	LANL
Application	Workload	Lublin model
	Number of jobs	1500
	Jobs I/O	10 MB
	Checkpoint delay	100 $ms - 5$ s

To include the workload for our simulation, the Lublin workload model (Lublin and Feitelson, 2003) had been used that generated 1500 jobs. The Lublin model specifies the number of processor required, arrival time of the jobs and time required (μ) for the job execution. The model correlates between job size, job running time, and job inter-arrival times for daytime cycles. The parameters used for Lublin workload model are given in Table 2. To inject a failure into resources we have used the Los-Alamos National Lab (LANL) failure traces[3]. It is one of the largest high performance computing sites and has a record of the past nine years. Failure frequencies had been modeled according to the Weibull distribution[4] while the mean repair time had been modeled according to a logarithmic distribution. Each site has different failure and repair ranges roughly spanning between an hour and a week.

[3]http://fta.inria.fr/apache2-default/pmwiki/index.php

[4]http://www.weibull.com/AccelTestWeb/weibull_distribution.htm

MRFCP evaluation

In this section, we are comparing three state of the art checkpointing techniques - i.e. PeriodicCP, LastFailureCP (Chtepen et al., 2009) and MeanFailureCP (Chtepen et al., 2009) - with the MRFCP. The PeriodicCP is a static checkpointing technique in which checkpoints are assigned according to a prior selection of time interval. In the LastFailureCP, the difference between the last failure time of a resource and the current system time is compared with the execution time of the job on a resource. While in the MeanFailureCP, the remaining time of a job is compared with the mean failure time of resource.

1. **Average job execution time:** The average job execution time of a job is also called its turnaround time. Fig. 3 plots the average job execution time, under various check-pointing techniques, against five different check-pointing intervals. In general, the average turnaround time increases while decreasing the checkpoint interval and vice versa. This may be attributed to the fact that more frequent checkpointing results in more network delay because of the overheads involved in requesting, storing and retrieving the checkpoints. Among the techniques, the PeriodicCP has the highest average turnaround time due to its static nature as it does not work on finding out the system load and relies on the manual configuration of checkpoint intervals. In comparison, the LastFailureCP omits unnecessary checkpoints and performs better if the system has not worked for more time, lastly. In other words, the LastFailureCP technique is totally dependent on the failure frequency of the resource. With the MeanFailureCP and MRFCP the average turnaround time is lesser, mainly due their dynamicity; the MeanFailureCP remaining, more or less, independent of the checkpoint interval. Not that, like LastFailureCP, the MeanFailureCP strategy also depends on the failure frequency of the resource and the average job execution time escalates, when the failure interval of a resource increases. MRFCP has the least average job execution time because it soaks up

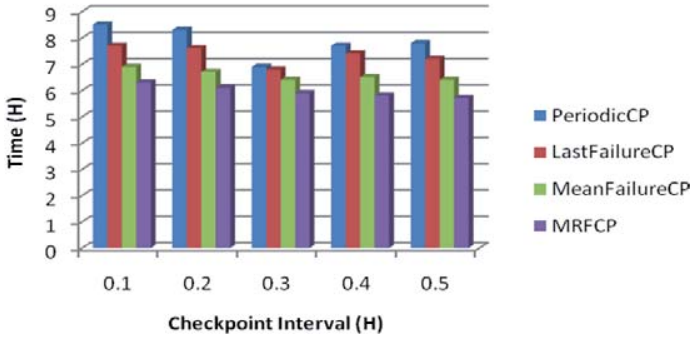

Figure 3: The average turnaround time for various check-pointing intervals

higher failure intervals of the resources. The results are even better for MRFCP at larger checkpoint intervals, a fact that makes it ideal for executing larger jobs. Moreover, it has been observed that MRFCP is a better technique when a resource has random failure intervals.

2. **Successful job execution:**

   This parameter shows us how many jobs was successfully executed by a fault tolerance technique at a given checkpoint interval. It can be readily observed from Fig. 4 that the number of successful jobs varies according to different checkpoint intervals. In the simulation for this comparison, we have injected a fixed number of faults to the system to see the effect of fault tolerance techniques due to varying checkpoint intervals. It can be seen that both MeanFailureCP and MRFCP techniques performs better than the rest of the techniques because both rely on some measure of central tendency to enhance the checkpoint interval depending on the job's remaining time and resource stability. The PeriodicCP and LastFailureCP are independent of system load and while the former only performs checkpointing that has been initially scheduled, the latter looks to the last failure occurrence of the resource. MRFCP starts producing better results as we move further in our checkpoint interval enhancement. This happens because it commences executing longer jobs too after the dilation of the checkpoint intervals thus leading to lower network delay and reduced overhead of the system.

3. **Average number of checkpoints:**

   Fig. 5 plots the average number of checkpoints as a function of varying checkpoint interval for various fault tolerance strategies. Typically, the number of checkpoints increases with decreasing checkpoint interval and vice versa. A better strategy is to consider taking the number of checkpoints dynamically according to the system load. The graph shows that PeriodicCP over-checkpoints at lower intervals, in comparison to the rest of the strategies, resulting in more overhead and escalated execution times. The reason is that the static checkpointing had been carried out in favor of higher intervals; that is why we see its best result at a checkpoint interval of $5\ H$. In such an eventuality the outcome is improved if the number of resource failures is less and the jobs have low execution time intervals. Generally, with MRFCP, the number of checkpoints are reduced as a function of the increasing checkpoint interval because on one hand the elongation of the interval gives more chance of processing to jobs with high execution times and, on the other, it performs processing while looking both at the resource failure occurrence and the remaining time of the jobs.

5 CONCLUSION

The MRFCP has demonstrated interesting results when compared with other state of the art checkpointing techniques. It showed better turnaround time and executed more jobs at any given checkpoint interval. Besides, it exhibited optimal checkpoints with respect to some fixed checkpoint interval. We believe that these results can be ameliorated further if one borrows something from the replication paradigm. The need is to devise a hybrid strategy that can adjust itself between replication and checkpointing. In addition, the simulations need to be conducted with some real experimental data - from Condor, for example, a scheduler that support check-pointing. The ultimate is to test the strategy, in combination with a good replication scheme, in a real environment, such as those briefly outlined in Xhafa et al. (2011).

REFERENCES

Agarwal, S., Garg, R., Gupta, M. S., and Moreira, J. E. (2004). Adaptive incremental checkpointing for massively parallel systems. In *Proceedings of the 18th annual international conference on Supercomputing*, ICS '04, pages 277–286, New York, NY, USA. ACM.

Bouabache, F., Herault, T., Fedak, G., and Cappello, F. (2008). Hierarchical replication techniques to ensure checkpoint storage reliability in grid environment. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster*

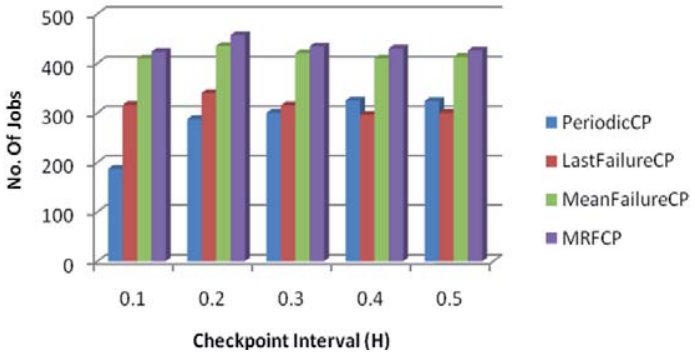

Figure 4: The number of successful job executions

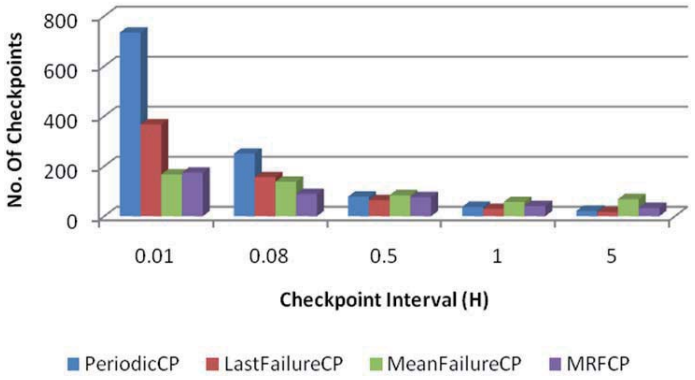

Figure 5: Average checkpoints at various intervals

Computing and the Grid, pages 475–483, Washington, DC, USA. IEEE Computer Society.

Bouguerra, M.-S., Gautier, T., Trystram, D., and Vincent, J.-M. (2010). A Flexible Checkpoint/Restart Model in Distributed Systems. In *Proceedings of the 8th international conference on Parallel processing and applied mathematics: Part I*, PPAM'09, pages 206–215, Berlin, Heidelberg. Springer-Verlag.

Bouguerra, M. S., Kondo, D., and Trystram, D. (2011). On the Scheduling of Checkpoints in Desktop Grids. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 305–313, Washington, DC, USA. IEEE Computer Society.

Buyya, R. and Murshed, M. M. (2002). Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CoRR*, cs.DC/0203019.

Cao, G. and Singhal, M. (2003). Checkpointing with mutable checkpoints. *Theor. Comput. Sci.*, pages 1127–1148.

Chtepen, M., Claeys, F. H. A., Dhoedt, B., De Turck, F., Demeester, P., and Vanrolleghem, P. A. (2009). Adaptive task checkpointing and replication: Toward efficient fault-tolerant grids. *IEEE Trans. Parallel Distrib. Syst.*, 20:180–190.

Deng, Y. and Park, E. (1994). Checkpointing and rollback-recovery algorithms in distributed systems. *Journal of Systems and Software*, 25(1):59 – 71.

Elnozahy, E. and Zwaenepoel, W. (1992). Manetho: transparent roll back-recovery with low overhead, limited rollback, and fast output commit. *Computers, IEEE Transactions on*, 41(5):526 –531.

Johnson, B. W. (1996). *An introduction to the design and analysis of fault-tolerant systems*, pages 1–87. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Koo, R. and Toueg, S. (1987). Checkpointing and rollback-recovery for distributed systems. *IEEE Trans. Softw. Eng.*, 13:23–31.

Lublin, U. and Feitelson, D. G. (2003). The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J. Parallel Distrib. Comput.*, 63:1105–1122.

Narasimhan, N., Moser, L., and Melliar-Smith, P. (2000). Transparent consistent replication of java rmi objects. In *Distributed Objects and Applications, 2000. Proceedings. DOA '00. International Symposium on*, pages 17 –26.

Plank, J. S., Beck, M., and Kingsley, G. (1995). Compiler-assisted memory exclusion for fast checkpointing. *IEEE TECHNICAL COMMITTEE ON OPERATING SYSTEMS AND APPLICATION ENVIRONMENTS*, 7:62–67.

Pruyne, J. and Livny, M. (1996). Managing checkpoints for parallel programs. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 140–154, London, UK. Springer-Verlag.

Ratner, D., Reiher, P., and Popek, G. (1999). Roam: a scalable replication system for mobile computing. In *Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on*, pages 96 –104.

Saito, Y. and Levy, H. M. (2000). Optimistic replication for internet data services. In *Proceedings of the 14th International Conference on Distributed Computing*, DISC '00, pages 297–314, London, UK, UK. Springer-Verlag.

Silva, L. and Silva, J. (1998). An experimental study about diskless checkpointing. In *Euromicro Conference, 1998. Proceedings. 24th*, volume 1, pages 395 –402 vol.1.

Tamir, Y. and Squin, C. H. (1984). Error recovery in multicomputers using global checkpoints. In *In 1984 International Conference on Parallel Processing*, pages 32–41.

Wong, K. F. and Franklin, M. (1996). Checkpointing in distributed computing systems. *Journal of Parallel and Distributed Computing*, 35(1):67 – 75.

Xhafa, F., Pllana, S., Barolli, L., and Spaho, E. (2011). Grid and P2P Middleware for Wide-Area Parallel Processing. *Concurrency and Computation: Practice and Experience*, 23(5):458–476.

AUTHOR BIOGRAPHIES

Suleman Khan is currently working as Senior Officer Network Maintenance in Pakistan International Airlines. He has done MS-Computer Science from CIIT, Abbottabad, Pakistan in 2011. He received his M.Sc computer Science from the Department of Computer Science, University of Peshawar, Pakistan in 2007. His research interests include resource management and job scheduling in grid, parallel and distributed computing as well as the wirelesssensor networks.

Khizar Hayat joined CIIT Abbottabad in Dec. 2009 and currently he is working as Associate Professor in the Computer Science Department. He received his PhD degree from the University of Montpellier II (UM2) in France, while working at the Laboratory of Informatics, Robotics and Microelectronics Montpellier (LIRMM). His areas of interest are image processing and information communication and security.

Sajjad Ahmad Madani works at COMSATS institute of information technology (CIIT) Abbottabad Campus as associate professor. He joined CIIT in August 2008 as Assistant Professor. Previous to that, he was with the institute of computer technology from 2005 to 2008 as guest researcher where he did his PhD research. Prior to joining ICT, he taught at COMSATS institute of Information Technology for a period of two years. He has done MS in Computer Sciences from Lahore University of Management Sciences (LUMS), Pakistan with excellent academic standing. He has already done BSc Civil Engineering from UET Peshawar and was awarded a gold medal for his outstanding performance in academics. His areas of interest include low power wireless sensor network and application of industrial informatics to electrical energy networks. He has published more than 35 papers in international conferences and journals.

Samee Ullah Khan is Assistant Professor of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA. Prof. Khan has extensively worked on the general topic of resource allocation in autonomous heterogeneous distributed computing systems. As of recent, he has been actively conducting cuttingedge research on energy-efficient computations and communications. A total of 111 (journal: 40, conference: 51, book chapter: 12, editorial: 5, technical report: 3) publications are attributed to his name. For more information, please visit: http://sameekhan.org/.

Joanna Kołodziej graduated in Mathematics from the Jagiellonian University in Cracow in 1992, where she also obtained the PhD in Computer Science in 2004. She joined the Department of Mathematics and Computer Science of the University of Bielsko-Biaa as an Assistant Professor in 1997. She has served and is currently serving as PC Co-Chair, General Co-Chair and IPC member of several international conferences and workshops including PPSN 2010, ECMS 2011, CISIS 2011, 3PGCIC 2011, CISSE 2006, CEC 2008, IACS 2008-2009, ICAART 2009-2010. Dr Koodziej is Managing Editor of IJSSC Journal and serves as a EB member and guest editor of several peer-reviewed international journals.