A Review on Evaluation of Multilevel Checkpointing System in Distributed Environment

Mr. Pratiek R Suraana Prof. Naresh Thoutam

Abstract — **Nowadays there is need of high performance of computer system in distributed environment. As the system mean time before failure correspondingly drops, applications must checkpoint frequently to make progress. However, at scale, the cost of checkpointing becomes prohibitive. A solution to this problem is multilevel checkpointing, which employs multiple types of checkpoints in a single run. Lightweight checkpoints can handle the most common failure modes, while more expensive checkpoints can handle severe failures. Also uses the designed of multilevel checkpointing library, the Scalable Checkpoint/Restart (SCR) library[1], that writes lightweight checkpoints to node-local storage in addition to the parallel file system, which present probabilistic Markov models of SCRs performance. The proposed work focuses on evaluation of multiple checkpointing in the distributed environment in the presence of multiple senders and multiple receiver.**

Key Words — **Checkpoint, Scalable Checkpoint/Restart, distributed environment**

I. Introduction

Although supercomputing systems use high quality components, they become less reliable at larger scales because increased component counts increase overall fault rates. HPC applications can encounter mean times between failures (MTBFs) of hours or days due to hardware breakdowns [1] and soft errors [16]. For example, the 100,000 node BlueGene/L system at Lawrence Livermore National Laboratory (LLNL) experiences an L1 parity error every eight hours [20] and a hard failure every 7-10 days. Exascale systems are projected to fail on the order of minutes or hours [14], [13], [10]. Most applications tolerate failures by periodically saving their state to reliable storage checkpoint files. Upon failure, an application can restart from a prior state by reading in a checkpoint. Checkpointing to a parallel file system is expensive at large scale. A single checkpoint can take tens of minutes [2], [12]. Further, large-scale computational capabilities have increased more quickly than I/O bandwidths. Typically, the limited bandwidth results from system design choices that optimize for system maintainability and availability. Increasing failure rates due to increases in system scale require more frequent checkpoints. Increased system imbalance makes them more expensive. Multilevel checkpointing, [37] uses multiple types of checkpoints that have different levels of resiliency and cost in a single application run to address this problem. The slowest but most resilient level writes to the parallel file system, which can withstand an entire system failure. Thus, an application can usually recover from a less resilient checkpoint level, given carefully chosen redundancy schemes. Multilevel checkpointing allows applications to take frequent inexpensive checkpoints and less frequent, more resilient checkpoints, resulting in better effciency and reduced load on the parallel file system.[1]

[1] evaluate multilevel checkpointing in large-scale systems through a probabilistic Markov model.

The major focus is on Details of Markov model of multilevel checkpointing, An extension of model for checkpointing to the parallel file system only upon job termination (checkpoint scavenging), An evaluation of the viability of checkpoint scavenging.

Overall, results demonstrate that multilevel checkpointing significantly improves current methods. We show that it can increase system efficiency significantly, with gains up to 35 percent while reducing the load on the parallel file system by a factor of two

II. Related Work

[40] shows that Reliability of digital systems can be improved through the use of redundant components. This paper is concerned with system failures caused by permanent component failures, in contrast to the problem of transient failures caused by noise. N+1parity was proposed by plank as way to perform diskless checkpointing, but the proposed James [32] algorithm is non incremental, and needs each processor to maintain two in memory copies of each check-point. In this [32], a set of checkpointing algorithm that perform no writing to disk, instead they assume that no more than m processors fail in a parallel or distributed system at any one time, and describe how to recover from such failures.Vaidya[31] shows the advantages of multi-level recovery schemes.

A multi-level recovery scheme is one that can tolerate different number of failures requiring larger costs. A single failure can be tolerated by rolling the system back to 1-checkpoint, while multiple recovery is possible by rolling back to an N-checkpoint. In this vaidya [30] shows that, to demonstrate the advantages of two-level recovery, we evaluate the performance of a recovery scheme that takes two different types of checkpoints, namely, 1-checkpoints and N-checkpoints.

A single failure can be tolerated by rolling the system back to a 1-checkpoint, while multiple failure recovery is possible by rolling back to an N-checkpoint. Gustavo etl. [29], selects a metric for the analysis and benchmarking of checkpointing

algorithms through simulation and provide evidence that this is an effective indicator of the overhead imposed by the checkpointing algorithm on distributed applications. [24] presents that, to tolerate failures in cluster and parallel computing systmes, parallel applications typically instru-
ment themselves with the ability to checkpoint their computation state to stable storage. When one or more processors fail, the application may be restarted from the most recent checkpoint, thereby reducing the amount of recomputation that must be performed. If any processor fails, a replacements processor is selected to take place of failed processor, and then all processors restore the saved state of the computation from the checkpoint. If no replacement processor
is available, then tha application is halted until one becomes avail-
able. The technique[21] of checkpointing and rollback recovery is a
well-known method to achieve fault tolerance in distributed computing system. In case of fault, the system can rollback to a consistent global state and resume computation without requiring additional efforts from the programmer. Checkpoint is a snapshot of the current state of process. It saves enough information in non-volatile stable storage, such that one can reconstruct the process state from the saved information. Messages received during the rolled back period, may also cause problem. Their sending processes will have no idea that these messages are to be sent again. Such a message whose send event is recorded in the state of sender process but the receive event is lost, is called a missing message [21]. Early in the deployment of the[20] Advanced Simulation and Computing (ASC) Q supercom-
puter, a higher-than-expected number of single node failures was observed. The elevated rate of single-node failures was hypothesized to be caused primarily by fatal soft errors. Error correction code (ECC) may be used to detect and correct soft errors.
[18]This paper examines methods of approximating the optimum restart strategy for minimizing application run time on a system exhibiting single component failures. This[16] paper analyze that, the data has been collected over the past 9 years at Los Alamos National Laboratory and includes 23000 failures recorded on more than 20 di_erent systems and perform analysis of failure as well as analysis of repair times. Glosli [14] , main focus is on parity error recovery methods BlueGene/L provides two methods to mitigate the
impact of L1 parity errors. The _rst option is to write through the L1 data cache directly to lower levels of the memory subsystem. The second option transfers control to an application. Activating write-through mode is very e_ective at eliminating parity errors[14].
[9] Multi-level checkpointing allows applications to take frequent inexpensive checkpoints and less frequent, more resilient checkpoints, resulting in better e_ciency and reduced load on the parallel file system. In this [3], Bland etl. presented an original scheme to enable forward recovery using only features of the current MPI standard.
Rollback recovery, which relies on periodic checkpointing has a variety of issues. The ideal period of checkpoint, a critical parameter, is particularly hard to assess. All these issues are overcome in [1], High- performance computing (HPC) systems are growing more powerful by utilizing more components. As the system mean time before failure correspondingly drops, applications must checkpoint frequently to make progress. However, at scale, the cost of checkpointing becomes prohibitive. A solution to this problem is multilevel check-pointing, which employs multiple types of checkpoints in a single run. Lightweight checkpoints can handle the most common failure modes, while more expensive checkpoints can handle severe failures.

III. Analysis

3.1 Multilevel checkpoint model

Our novel probabilistic model of multilevel checkpointing can predict the behavior of SCR given the factors that can affect its performance. We evaluate SCR's performance using the model with parameters that represent SCR usage by pF3D. This model can guide general use of multilevel checkpoint systems for current and future systems and motivate system designs that provide adequate overall reliability and efficiency. To model multilevel checkpointing systems, we make some simplifying assumptions that naturally introduce errors into the model's predictions. However, these errors are relatively small. We now discuss our assumptions and their potential impact.
We assume that failures are independent. Thus, a failure within a job does not increase the probability of another failure within that job or future jobs. In reality, some failures are correlated. However, SCR is designed to mitigate effects of correlated failures. For example, it can avoid using failed nodes in a job allocation as those nodes may be likely to fail again.
We assume that checkpoints are globally coordinated and taken at regular intervals throughout the job. While not always true, SCR requires globally coordinated checkpoints and pF3D does checkpoint at regular intervals. We also assume costs to read and write checkpoints are constant throughout the job. However, read and write times actually vary, particularly when shared resources such as the parallel file system are used, which leads to some error in our model. We assume that the application recovers from the most recent viable checkpoint when a failure occurs. We do not model possible savings from using an older checkpoint that is also sufficient for recovery but available from faster storage. Thus, we may underestimate the possible performance of multilevel checkpointing.
We assume an infinite pool of spare nodes. When using SCR in practice, users often request extra nodes in their job

allocation; upon node failure SCR restarts the job using the extra nodes, ignoring those that failed. In general, the failure rate is less than the repair rate, so this assumption typically holds. In the absence of spare nodes, SCR copies the most recent checkpoint to the parallel file system and terminates the job; we model this capability in Section 6. Similarly, the model does not account for batch system allocation time limits.

We assume a single level L checkpoint period completes within the allocation time limit. In practice, SCR handles batch limits by copying the most recent checkpoint to the parallel file system before the allocation expires. These assumptions cause our model to overestimate performance if SCR must copy checkpoints to the parallel file system.

3.2 Model Overview

In a multilevel checkpointing system, each of L checkpointing mechanisms is a level, for which level 1 checkpoints are the least expensive and resilient, while level L checkpoints are the most expensive and resilient. In our model, we assume that a checkpoint at level k can be used to recover from a superset of the failure modes that are recoverable using checkpoints at levels less than *k*. A level *k* failure refers to a failure severe enough that we require a checkpoint at level $i \geq k$ for recovery. A level *k* recovery restores an application using a checkpoint saved at level *k*. A multilevel checkpointing system alternates between different types of checkpoints. Since more severe failures happen less frequently, the system records zero or more level k checkpoints for every level *k+1* checkpoint.

In our Markov model of a multilevel checkpointing system, nodes represent application states and edges represent the transitions between states. We annotate each edge with the probability that the application will transition from the source state to the destination state and with cost information such as the time spent in the source state given that the transition is taken. Kathryn says that model has computation and recovery states. Computation states represent periods of application computation followed by a checkpoint. Recovery states represent the process of restoring an application from a checkpoint saved previously.

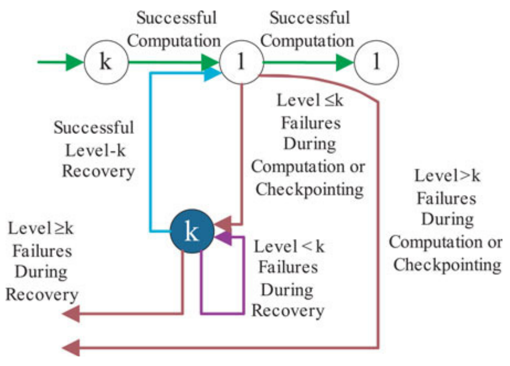
Successful Computation
Successful Computation
k
1
1
Successful Level-k Recovery
Level ≤k Failures During Computation or Checkpointing
Level>k Failures During Computation or Checkpointing
k
Level ≥k Failures During Recovery
Level < k Failures During Recovery

Fig 3.1: Basic structure of multilevel Markov model.

Fig 3.1 presents our model's basic structure. The white states in the top row are computation states, and the single blue state at the bottom is a recovery state. We label each computation state by the checkpoint level with which it terminates and the recovery state by the checkpoint level that it uses to restore the application. If no failures occur during application execution or checkpointing, the application transitions from one computation state to the next. If a failure occurs, the application transitions to the recovery state corresponding to the most recent checkpoint capable of recovering from the failure. For example, if a failure at level *i* and $i \leq k$ while in the middle computation state in Fig 3.1, the system transitions to recovery state *k*, which restores the application using the checkpoint that was written at the end of the previous computation state. However, if $i \geq k$, the system must transition to a recovery state that corresponds to an older checkpoint saved at a higher level.

If no failures occur during recovery, the application transitions to the computation state that follows the checkpoint used for recovery. If a failure at level $i < k$ occurs while in a level *k* recovery, we assume the current recovery state must be restarted. However, if $i \geq$ k, the application must transition to a higher-level recovery state. We assume a level *L* recovery can be restarted to recover after a failure at any level.

We exploit the recursive structure of our model to develop recurrence equations that we efficiently solve for the expected run time.

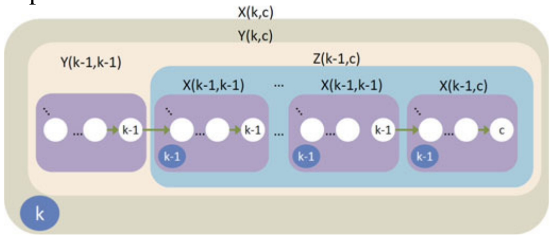

Fig 3.2: Hierarchical structure of Markov model.

As Fig 3.2 shows, we can build a full model by recursively composing three basic blocks, which we label *X (k,c), Y(k,c), and Z(k,c)*, where $k,c \in 1, 2, \ldots, L$.

An *X (k,c)* block consists of a *Y(k,c)* block and a base state for recovery at level *k*, R_k. A *Z(k,c)* block consists of a series of *X(k,k)* blocks and a terminating *X(k,c)* block.

When $k > 1$, a *Y(k,c)* block consists of either a single *Y(k-1,c)* block or a *Y(k-1,k-1)* block followed by a *Z(k-1,c)* block. Finally, when $k=1$, a $Y(k=1,c)$ block is a base state corresponding to a computation state that terminates with a checkpoint at level *c*. The parameter *c* is the checkpoint level taken by the last compute state in a block and *k* is the level of a block. An instance of *X(L,L)* represents a level *L* interval.

3.3 Base States

For the base computation and recovery states, p_0 is the probability that the application executes for some time, t_0, without encountering a failure. For $k \in 1,2,\dots.L$, the probability that the first failure during this period occurs at level k is p_k and t_k is the expected run time before encountering that failure. With T representing the time for spent in the state before exiting, and assuming an exponential distribution, the expressions for $p_0(T)$ and $t_0(T)$ evaluate to $p_0(T) = e^{-\lambda T}$ and $t_0(T) = T$, and for $k \in 1,2,\dots.L$, $p_k(T)$ and $t_k(T)$ evaluate to

$$p_k(T) = \lambda_k / \lambda\ (1 - e^{-\lambda T}),$$

$$t_k(T) = 1 - (\lambda T+1).e^{-\lambda T} / \lambda\ .(1 - e^{-\lambda T}),$$

where $\lambda = \lambda_1 + \lambda_2 + \dots\dots + \lambda_L$.

A $Y(k = 1, c)$ block is a base computation state in which the application executes for an interval of length t and then writes a checkpoint at level c, which requires a time of c_c. From the formulas above, $p_{Y0} = p_o\ (t+c_c)$ and $t_{Y0} = t_0(t+c_c)$, and for $i \in 1,2,\dots,L$, $p_{Yi} = p_i(t+c_c)$, and when $p_{Yi} = t_i(t+c_c)$.

Table 1
Model Parameters

Symbol	Definition
L	Number of checkpoint levels modeled
υ_k	Number of level k checkpoints within each level k+1 period
T	Length of compute interval before the application initiates a checkpoint
c_k	Time to record a level k checkpoint
r_k	Time to complete a level k recovery
λ_k	Average rate of level k failures assuming Poisson distributions

[1] use the definitions listed in Table 1 to parameterize the multilevel checkpoint model.

While in a recovery base state at level k, the system is recovering from a failure using a checkpoint saved at level k, which requires a time of r_k. We find that the probability of exiting with no failures is $p_{R0} = p_0(r_k)$ and the time to exit with no failures is $t_{R0} = t_0(r_k)$. For $i \in 1,2,\dots\dots,L$, the probability of exiting on a failure at level i is $p_{Ri} = p_i(r_k)$, when $p_{Ri} > 0$, the time before exiting on failure at level i is $t_{Ri} = t_i(r_k)$.

IV. Performance Analysis

4.1 The X(k,c) Block

An $X = X(k,c)$ block internally consists of a $Y = Y(k,c)$ block and a recovery state at level k, $R = R_k$. To simplify the final expressions, we merge groups of related transitions into single transitions. We show the merged transitions in Fig. 3.

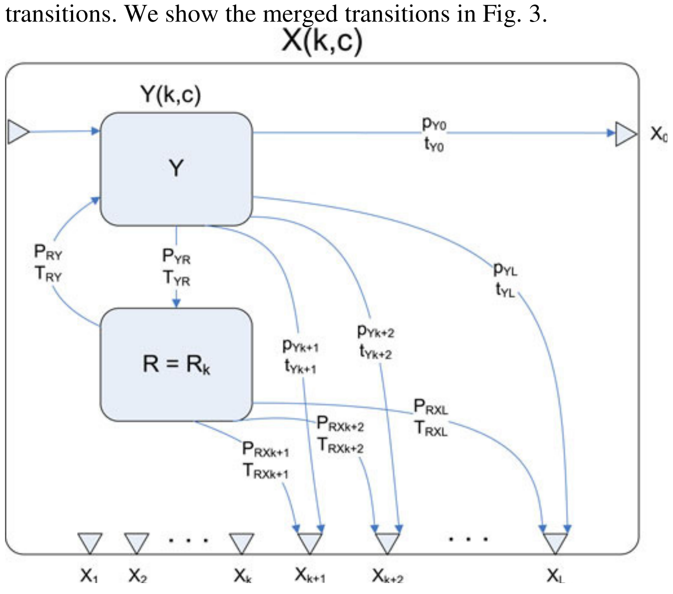

Fig 4.1: Simplified diagram of X(k,c)

Y transitions to the recovery state R for any failure scenario that requires a recovery level at k or less. We merge each of these transitions into a single transition that has probability of P_{YR} and an expected run time of T_{YR}. Once in R, a transition away from R eventually happens, provided that $\Sigma^{k}_{i=0}\ p_{Ri} < 1$.
However, one or more loops back to R may occur before transitioning away. We merge the transitions from R to Y ; its probability is P_{RY} and its expected run time is T_{RY} as shown in Fig 4.1.
Failures at levels $i \geq k$ cause a transition out of R_k to a higher level recovery state. The transitions from R have probability P_{RXi} and expected run time T_{RXi} . However, if $k = L$, then recovery is restarted upon failure at any level so for each $i \in 1, 2, \dots L$, $P_{RXi} = 0$. While in a recovery state at level $k < L$, the system transitions to a recovery state at level $k + 1$ if a level k or level $k + 1$ failure occurs. Otherwise, for the occurrence of a failure at level i, where $i > k + 1$, a transition is made to a recovery state at level i.

4.2 The *Z(k,c)* Block

A $Z = Z(k,c)$; block only exists when $\upsilon_k > 0$. It consists of a chain of $X = X(k,k)$ blocks of length $\upsilon_k - 1$ followed by a $X' = X(k,c)$ block, as Fig 4.2 shows. Then define $\upsilon = \upsilon_k - 1$. The probability of successfully transitioning from the Z block to the first computation state of the next block is the probability that υ consecutive successful transitions from X blocks are followed by one successful transition from the X' block, $p_{Z0} = (p_{X0})^{\upsilon}\ .\ p_{X'0}$. When $p_{Z0} > 0$, the expected time to make this transition is $t_{Z0} = \upsilon\ .\ t_{X0} + t_{X'0}$ .The total probability to leave Z for a recovery state at level i is the sum of the probabilities corresponding to each of the possible paths from the substates of Z.

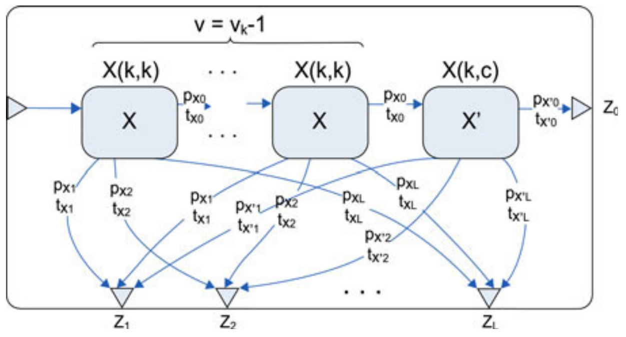

Fig 4.2 The *Z(k,c)* State.

4.3 The Y(k,c) Block

A *Y(k,c)* block is built using three different constructions depending on the values of *k* and (when $k > 1$) υ_{k-1} . If $k = 1$, then $Y(k,c) = Y(k=1,c)$, which is a base computation state. If $k > 1$ and $\upsilon_{k-1} = 0$, then $Y = Y(k,c)$ consists of a single $Y' = Y(k-1,c)$ block. We compute the probabilities and expected run times to transition from *Y* given the probabilities and expected run times to transition from *Y′* as $p_{Y0} = p_{Y'0}$ and $t_{Y0} = t_{Y'0}$, and , for each level $i \in 1,2,\ldots,L$, $p_{Yi} = p_{Y'i}$ and $t_{Yi} = t_{Y'i}$.

If $k > 1$ and $\upsilon_{k-1} > 0$, then $Y = Y(k,c)$ consists of a starting $Y' = Y(k-1, k-1)$ block followed by a $Z = Z(k-1, c)$ block, as Fig 4.3 shows.

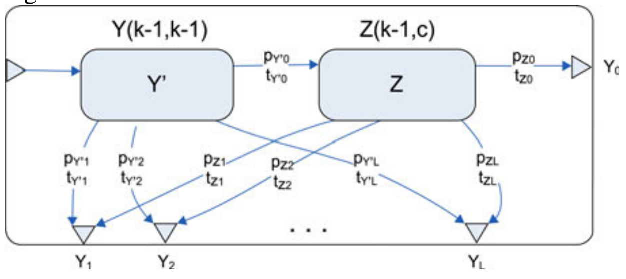

Fig 4.3. The *Y(k,c)* State for $k > 1$ and $\upsilon_{k-1} > 0$.

The probability that a successful transition from *Y* occurs is the probability that both *Y′* and *Z* transition successfully, $p_{Y0} = p_{Y'0} \cdot p_{Z0}$ and the expected time for this transition is $t_{Y0} = t_{Y'0} + t_{Z0}$. The total probability to leave *Y* for a recovery state at level i is the sum of the probabilities for each path.

Conclusion

Presents the models of multilevel checkpointing with SCR, which we validated against results from executions and simulations. Our novel, hierarchical Markov models predict the performance of multilevel checkpointing systems based on system reliability and checkpoint cost. Analysis demonstrates that multilevel checkpointing significantly improves system efficiency, particularly as failure rates and relative parallel file system checkpoint costs increase.
[1] explored the impact of checkpoint scavenging, a key extension to multilevel checkpointing.

Acknowledgement

The authors would like to acknowledge Computer Engineering department, SITRC and all the people who provided with the facilities being required and conductive conditions for completion of the review paper.

References

[1] Kathryn Mohror, Adam Moody, Greg Bronevetsky, and Bronis R. de Supinski, Member, IEEE Computer Society, " Detailed Modeling and Evaluation of a Scalable Multilevel Checkpointing System ", IEEE transactions on parallel and distributed systems, VOL. 25, NO. 9, September 2014
[2] J. Daly, et al., Inter-Agency Workshop on HPC Resilience atExtremeScale, http://institutes.lanl.gov/resilience/docs/Inter- Agency ResilienceReport.pdf, Feb. 2012.
[3] W. Bland, P. Du, A. Bouteiller, T. Herault, G. Bosilca, and J.Dongarra, A Checkpoint-on-Failure Protocol for Algorithm-Based Recovery in Standard MPI, Proc. 18th Intl Conf. Parallel Processing (Euro-Par12), C. Kaklamanis, T. Papatheodorou, and P. Spairakis, eds., pp. 477-488, Springer, 2012.
[4] P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra, "Algorithm-based Fault Tolerance for Dense Matrix Factorizations," in 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming . ACM, 2012.
[5] L.A. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka, FTI: High Performance Fault Tolerance Interface for Hybrid Systems, Proc. Intl Conf. for High Performance Computing, Networking, Storage and Analysis (SC 11), 2011.
[6] J. Dongarra, P. Beckman et al., The international exascale software roadmap," IJHPCA, vol. 25, no. 11, pp. 3-60, 2011.
[7] F. Cappello, H. Casanova, and Y. Robert, Preventive migration vs. preventive checkpointing for extreme scale supercomputers," PPL 21:2 , pp. 111-132, 2011
[8] T. Davies, C. Karlsson, H. Liu, C. Ding, , and Z. Chen, "High Performance Linpack Benchmark- A Fault Tolerant Implementation without Checkpointing," in Proceedings of the 25th ACM International Conference on Supercomputing (ICS 2011) . ACM
[9] A. Moody, G. Bronevetsky, K. Mohror, and B.R. de Supinski, Design, Modeling, and Evaluation of a Scalable Multi-level Check-pointing System, Proc. Intl Conf. for

High Performance Computing, Networking, Storage and Analysis (SC10), pp. 1-11, Nov. 2010.
[10] E. Vivek Sarkar, ed., ExaScale Software Study: Software Challenges in Exascale Systems, 2009.
[11] F. Cappello, A. Geist, B. Gropp, L. V. Kale, B. Kramer, and M. Snir, Toward exascale resilience," IJHPCA , vol. 23, no. 4, pp. 374-388, 2009.
[12] K. Iskra, J.W. Romein, K. Yoshii, and P. Beckman, ZOID: I/O Forwarding Infrastructure for Petascale Architectures, Proc. 13th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP 08), pp. 153-162, 2008
[13] K. Bergman, et al., ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, http://citeseerx.ist.- psu.edu/view doc/summary?doi=10.1.1.165.6676, 2008.
[14] J.N. Glosli, K.J. Caspersen, J.A. Gunnels, D.F. Richards, R.E. Rudd, and F.H. Streitz, Extending Stability Beyond CPU Millennium: A Micron-Scale Atomistic Simulation of Kelvin-Helmholtz Instability, Proc. ACM/IEEE Conf. Supercomputing (SC 07), pp. 1-11, 2007.
[15] B. Schroeder and G. A. Gibson, "Understanding Failures in Petascale Computers," SciDAC, Journal of Physics: Conference Series , vol. 78, 2007.
[16] B. Schroeder and G.A. Gibson, A Large-Scale Study of Failures in High-Performance Computing Systems, Proc. Intl Conf. Dependable Systems and Networks (DSN 06), pp. 249-258, June 2006.
[17] R. Ross, J. Moreira, K. Cupps, and W. Pfei_er, Parallel I/O on the IBM Blue Gene/L System,Blue Gene/L Consortium Quarterly Newsletter, First Quarter, 2006.
[18] J. Daly, A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps, J. Future Generation Computer Systems, vol. 22, no. 3, pp. 303-312, http://www.sciencedirect.com/science/article/B6V06-F490KH-6/2/6ebfa65591e5d0eb09e2ae5ae3b2ed44, 2006.
[19] J. T. Daly, A higher order estimate of the optimum checkpoint interval for restart dumps,"Future Gener. Comput. Syst. , vol. 22, pp. 303-312, February 2006.
[20] S.E. Michalak, K.W. Harris, N.W. Hengartner, B.E. Takala, and S.A. Wender, Predicting the Number of Fatal Soft Errors in Los Alamos National Laboratorys ASC Q Supercomputer, IEEE Trans. Device and Materials Reliability, vol. 5, no. 3, pp. 329-335, Sept. 2005.
[21] Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra, Fault tolerant high performance computing by a coding approach," in Proceedings of the tenth ACM SIG- PLAN symposium on Principles and practice of parallel programming , ser. PPoPP '05. New York, NY, USA: ACM, 2005, pp. 213-223
[22] W. Gropp and E. Lusk, Fault tolerance in message passing interface programs," Int. J. High Perform. Comput. Appl. , vol. 18, pp. 363-372, August 2004.
[23] B.S. Panda and S.K. Das, Performance Evaluation of a Two Level Error Recovery Scheme for Distributed Systems, Proc. Fourth IntlWorkshop on Distributed Computing, Mobile and Wireless Computing (IWDC 02), pp. 88-97, 2002
[24] J.S. Plank and M.G. Thomason, Processor Allocation and Check-point Interval Selection in Cluster Computing Systems, J. Parallel Distributed Computing, vol. 61, no. 11, pp. 1570-1590, 2001.
[25] J. S. Plank and M. G. Thomason, "Processor allocation and checkpoint interval selection in cluster computing systems," JPDC , vol. 61, p. 1590, 2001.
[26] G. Fagg and J. Dongarra, "FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world," EuroPVM/MPI , 2000.
[27] L. Silva and J. Silva, Using Two-Level Stable Storage for Efficient Checkpointing, Proc. IEEE Software, vol. 145, no. 6, pp. 198-202, Dec. 1998.
[28] R.L. Berger, C.H. Still, E.A. Williams, and A.B. Langdon, On the Dominant and Subdominant Behavior of Stimulated Raman and Brillouin Scattering Driven by Nonuniform Laser Beams, Physics of Plasmas, vol. 5, pp. 4337-4356, 1998.
[29] J. Dongarra, L. Blackford, J. Choi et al. , ScaLAPACK user's guide," Society for Industrial and Applied Mathematics, Philadel-phia, PA , 1997.
[30] N.H. Vaidya, A Case for Two-Level Distributed Recovery Schemes, Proc. ACM SIGMETRICS Joint Intl Conf. Measurement and Modeling of Computer Systems (SIGMETRICS 95), pp. 64-73, 1995.
[31] N.H. Vaidya, A Case for Multi-Level Distributed Recovery Schemes, Technical Report 94-043, Texas A and M University, May 1994.
[32] J.S. Plank and K. Li, Faster Checkpointing with N+1 Parity, Proc. 24th Intl Symp. Fault-Tolerant Computing (FTCS 94), pp. 288-297, June 1994.
[33] F. Luk and H. Park, An analysis of algorithm-based fault tolerance techniques," Journal of Parallel and Distributed Computing , vol. 5, no. 2, pp. 172-184, 1988.
[34] K. Huang and J. Abraham, Algorithm-based fault tolerance for matrix operations," Computers, IEEE Transactions on , vol. 100, no. 6, pp. 518-528, 1984.
[35] A. Duda, The E_ects of Checkpointing on Program Execution Time, Information Processing Letters, vol. 16, no. 5, pp. 221-229, 1983.
[36] E. Gelenbe, On the optimum checkpoint interval," JoACM , vol. 26, pp. 259-270,1979.
[37] E. Gelenbe, A Model of Roll-back Recovery with Multiple Check-points, Proc. Second Intl Conf. Software Engineering (ICSE 76), pp. 251-255, 1976.
[38] J.W. Young, A First Order Approximation to the Optimum Checkpoint Interval, Comm. ACM, vol. 17, no. 9, pp. 530-531, 1974.
[39] J. W. Young, A rst order approximation to the optimum check-point interval,"Commun. ACM , vol. 17, pp. 530-531, September 1974.
[40] R.E. Lyons and W. Vanderkulk, The Use of Triple-Modular Redundancy to Improve Computer Reliability, IBM

J. Research and Development, vol. 6, no. 2, pp. 200-209, 1962.

AUTHOR'S PROFILE

	Author's Name: **Mr. Pratiek Rajendrra Suraana** is a P.G. student of Computer Engineering at SITRC College of Engineering , Nasik under Savitribai Phule Pune University . He has completed her undergraduate Course of engineering from Savitribai Phule Pune University. He areas of interest include Computer Network.
	Author's Name : **Prof. Naresh Thoutam** Completed his M.Tech M.Tech(CSE) from Walchand College Of Engineering ,Sangli.And B.Tech(CSE) from JNTU KAKINADA.Presently he is working as Assistant Prodessor at SITRC College of Engineering, Nasik, Maharashtra, India.