Checkpointing to minimize completion time for Inter-dependent Parallel Processes on Volunteer Grids

Mohammad Tanvir Rahman Hien Nguyen Jaspal Subhlok Gopal Pandurangan
Department of Computer Science, University of Houston
Houston, TX 77204, USA
Email: mtrahman3@uh.edu, hxnguyen4@uh.edu, jaspal@uh.edu, gopal@cs.uh.edu

***Abstract*—Volunteer computing is being used successfully for large scale scientific computations. This research is in the context of Volpex, a programming framework that supports communicating parallel processes in a volunteer environment. Redundancy and checkpointing are combined to ensure consistent forward progress with Volpex in this unique execution environment characterized by heterogeneous failure prone nodes and interdependent replicated processes. An important parameter for optimizing performance with Volpex is the frequency of checkpointing. The paper presents a mathematical model to minimize the completion time for inter-dependent parallel processes running in a volunteer environment by finding a suitable checkpoint interval. Validation is performed with a sample real world application running on a pool of distributed volunteer nodes. The results indicate that the performance with our predicted checkpoint interval is fairly close to the best performance obtained empirically by varying the checkpoint interval.**

***Keywords*-Fault tolerant execution; checkpointing; volunteer computing; PC grids**

I. Introduction

The goal of the research presented in this paper is to minimize the completion time for inter-dependent parallel processes running in a volunteer environment. The pool of hosts in a volunteer environment consists of distributed computers which can be used for performing large scale computations when "idle". Here "idle" implies the conditions under which the owner of the machine allows his/her computer's use for volunteer computing. The computers can be geographically distributed and connected through the internet. At present, volunteer computing is being used successfully for large scale scientific computations, commonly employing BOINC [3]. Current large volunteer communities can provide computation power in the range of Petaflops. The parallel processes running on different volunteer clients can coordinate and communicate with each other with Volpex [8] which provides a custom-defined put/get model for communicating processes.

One of the key characteristics of volunteer computing is that the hosts are highly unreliable as they can leave the available pool simply because the owner turns them off or starts using them. Periodic checkpointing and maintaining redundant replicas are important for making continuous forward progress in a volunteer environment. In this paper, our goal is to minimize the overall completion time by predicting a suitable checkpoint interval in the presence of process replication. Hence, this research is important for effective use of volunteer nodes for large parallel applications. Our proposed solution is deployed and evaluated on a pool of volunteer nodes managed by BOINC and Volpex.

Creating checkpoints is an effective technique to deal with failures. Making checkpoints too frequently will waste time and resources, but on the other hand, making infrequent checkpoints will result in a significant amount of work-loss in case of client failures. The presence of redundant replicas of individual processes makes the optimal checkpoint interval calculation more complex. To the best of our knowledge, state-of-the-art in checkpoint optimization does not provide any solution for replicated coordinated parallel processes.

There is a large body of research on optimizing the checkpoint interval, some examples being [10], [7]. However, most of the earlier work focuses on finding the optimal checkpoint for a single process or independent distributed processes. Presence of communicating processes fundamentally changes the execution model as the slowdown or failure of a single process impacts the entire computation. Presence of replicated processes further complicates the execution model, as the computation can continue seamlessly despite failures as long as at least one replica of each process is alive. Current Volpex framework uses a heuristic predefined checkpoint interval for application execution that is based on the programmer's intuition. Work presented in this paper automates the process of checkpoint interval selection for multiple inter-dependent parallel processes, each possibly with multiple replicas. Our proposed mathematical model allows us to calculate the checkpoint interval that is optimal (under this model). Our experiments show that the checkpoint interval predicted by our analysis yields a job completion time that is close to the minimum.

To summarize, our main contributions are:

1) Designing a probability based mathematical model to predict a checkpoint interval with the lowest completion time for inter-dependent parallel processes with multiple replicas. The true optimal checkpoint interval is unknown and can only be estimated empirically.

2) Implementation and evaluation of the checkpoint prediction model in a real world volunteer computing framework with a real world application to validate our theoretical analysis.

II. Previous Work

Young [13] initiated the work on approximating the optimum checkpoint interval for minimizing application run time. He gave a simple mathematical equation to find the optimal checkpoint interval (T_c^{opt}) which is very easy to apply in practice. However, Young did not consider multiple parallel processes in his work. Our approach is very similar to Young's approach (for one process), but we do a simpler analysis, since we assume that when a process fails it has to wait till checkpointing to restart. In fact, when we use a crude (first-order) approximation, we get essentially the same formula as Young's approximation (but without the factor of $\sqrt{2}$) (see Section IV.A). The advantage of our approach is that it is simpler, and also leads to a more accurate equation for T_c^{opt} (under the assumptions) and generalizes easily for multiple processes and replicas.

Daly discussed three models for predicting the runtime and optimum restart interval [5]. He evaluated these models and used the results to derive a simple method for calculating the optimum restart interval. He further developed a higher order model for finding the optimum restart interval on a system exhibiting Poisson single component failures [4]. However, checkpoint interval for inter-dependent parallel processes are not discussed in Daly's work.

III. Execution Model

The research in this paper is motivated by the challenges faced in application execution under Volpex. We provide a brief introduction to the Volpex execution framework as that also forms the context for the results developed in this paper.

BOINC is a well-known middleware employed by Volpex for the basic control, node recruitment and management of execution for running scientific applications over volunteer nodes. Volpex provides the programming framework [9], [2] and runtime support for the execution of communicating parallel codes on volunteer nodes. The number of hosts to be recruited depends on the number of processes n, level of replication r, the number of requested spare hosts, and other user-defined parameters. During the execution of the application, new process instances can be created by selecting nodes from a pool of "hot spares". Additional nodes can be recruited during execution when the pool of spares is diminished to a preset value because of host failures.

Volpex can explicitly create process replicas during program invocation or a new instance may be created from a checkpoint to replace a slow or failed process instance. Process replicas are not aware of the existence of, or coordinate with, other process replicas and usually act autonomously. A process can checkpoint its state independently without the coordination of other processes. The system can recreate a process from a checkpoint irrespective of whether the original process is dead or alive, and without coordinating with other replicas or processes. Volpex uses "heartbeat monitoring" to detect process failure.

In a checkpointing Volpex application, each process instance makes a *StoreCheckpoint* request to the server after performing a fixed amount of work (for example, finishing a fixed number of loop iterations), rather than after a fixed time period. A fast client will send this request more frequently than a slower machine. The server determines if the *StoreCheckpoint* request will lead to the recording of an actual checkpoint. A *StoreCheckpoint* request normally leads to an actual recording of a checkpoint, if: (1) it represents the most recent checkpoint state, i.e., there does not already exist a checkpoint on the server that represents a process state later in logical time (this is possible with replicas) and (2) a specified *checkpoint interval* has elapsed since the recording of the last checkpoint. If these conditions are satisfied, the server will accept that *StoreCheckpoint* request and store a checkpoint on the server. Otherwise, the server will ignore the request.

Each application is configured to execute with a fixed number of replicas for each process. When all replicas of a process are dead (less common), a new process is created immediately from the most recent checkpoint on the server. When a process replica is dead but other replicas remain (more common), a new instance of the process is created after getting the next valid *StoreCheckpoint* request from another replica. By default, Volpex applications have to be configured with a heuristic fixed *checkpoint interval* for all the processes as input before the execution begins. The goal of this work is to determine the checkpoint interval automatically for applications that optimizes performance.

IV. A mathematical model and analysis

In this section, we develop the mathematical framework for predicting the optimal checkpointing interval. We assume a simple probabilistic model for process failures and analyze the optimal checkpoint interval under this assumption. Following are the input parameters:

1) Number of processes: n
2) Number of replicas for each process: r
3) Time to create a checkpoint: T_s. (See Section V.B for details)
4) Success probability distribution (of a single process with no replica): $p = f(t)$. This gives the probability p (which depends on t) to reach a checkpoint without failure. In other words, the probability of a process being alive till time t is $f(t)$. In this work, we will assume the success probability distribution to be the *exponential distribution* function [12]. The main

property of the exponential distribution is the *memorylessness* property: given that there is no failure till time t, the probability of failure till time $t+s$ is exactly the probability of failure till time s. An equivalent way is to assume that the occurrence of failures is a *Poisson* process with failure rate λ. The success probability distribution of the exponential distribution is defined as

$$f(t) = e^{-\lambda t} \tag{1}$$

where λ is the failure rate or $\frac{1}{\lambda}$ is the mean time to failure (MTTF).

Our goal is to compute T_c^{opt}, the optimum checkpoint interval. To do this we will compute the expected time incurred (under the given success distribution) to successfully reach a checkpoint (including the time to create a checkpoint). Let's denote this expected time by $G(T_c)$, which is a function of T_c. $G(T_c)$ can be considered the (expected) overhead associated with having checkpoint interval as T_c under success distribution p. The "normalized" overhead is $G(T_c)/T_c$, i.e., the ratio of the expected overhead time to the actual checkpoint interval (T_c is the time when "useful" work is done). This captures the amount of overhead incurred per checkpoint interval. The optimum checkpoint interval T_c^{opt} is the checkpoint interval that minimizes $G(T_c)/T_c$.

To simplify our analysis, we make the following assumptions. Validation of these assumptions are given in [11].

1) A faulty process can be restarted only after every checkpoint interval.
2) Time to detect fault and restart is negligible.
3) All the nodes in volunteer network follow the same success probability distribution. (Our experimental results validate this hypothesis.)

A. Analysis for single process with single replica

Assuming that the checkpoint interval is T_c, we calculate $G(T_c)$, the expected overhead.

Starting from time zero, after time T_c, the probability to reach the next checkpoint successfully is given by $p = f(T_c) = e^{-\lambda T_c}$. Since we assumed that a faulty process can be restarted only after every checkpoint interval period, if the process fails (which happens with probability $1-p$), we shall try to reach next checkpoint in time $2T_c$ time. If the process fails again, we shall try to reach the next checkpoint by time $3T_c$ and so on. Note that once we successfully reach the end of a checkpoint interval, we spend T_s time to create a checkpoint which contributes to the overhead.

From the above discussion, for a single process with no replica, the normalized overhead $G(T_c)/T_c$ is

$$\begin{aligned}\frac{G(T_c)}{T_c} &= \frac{1}{T_c}(p(T_c+T_s) + (1-p)p(2T_c+T_s) + \\ &\quad (1-p)^2 p(3T_c+T_s) + (1-p)^3 p(4T_c+T_s) + \ldots) \\ &= \frac{1}{f(T_c)} + \frac{T_s}{T_c} \text{ (replacing } p \text{ with } f(T_c)) \end{aligned} \tag{2}$$

To minimize the normalized overhead for a single process with no replica, we will differentiate equation 2 with respect to T_c, set the value to zero and replace $f(T_c) = e^{-\lambda T_c}$ (since we assume the exponential distribution). Please note T_s is a constant. After simplification and using WolframAlpha's [1] computational engines, we get the optimal checkpoint interval T_c^{opt} as follows:

$$T_c^{opt} = \frac{2W(\frac{\sqrt{\lambda T_s}}{2})}{\lambda} \tag{3}$$

where $W(z)$ is the Lambert W function. (The Lambert W function is the *inverse function* of the function $f(W) = We^W$.)

Using a first order approximation, $e^x \approx 1+x$, we can get a closed form formula for $T_c^{opt} = \sqrt{\frac{T_s}{\lambda}}$. Of course, using higher order approximations for the exponential function gives more accurate values of T_c^{opt}. Another option is to use compute W function using mathematical software such as Mathematica.

B. Generalization to multiple processes with many replicas

If we have n inter-dependent parallel processes and with r replicas for each process, the probability for all the n processes to successfully reach a checkpoint without failure = $(1-(1-p)^r)^n$. (More explanation can be found in [11].) This is because, each of the n processes will successfully reach a checkpoint, if at least one of the replicas of each process reaches a checkpoint.

Thus, replacing $f(T_c)^n$ by $(1-(1-p)^r)^n$ in eq: 2, we get, for n processes each with r replicas, the normalized expected overhead is:

$$\frac{G(T_c)}{T_c} = \frac{1}{(1-(1-p)^r)^n} + \frac{T_s}{T_c} \tag{4}$$

Using the same approach like Section IV.A we get the formula,

$$\frac{T_s}{T_c^2} = \frac{nr\lambda e^{-\lambda T_c}(1-e^{-\lambda T_c})^{r-1}}{(1-(1-e^{-\lambda T_c})^r)^{n+1}} \tag{5}$$

Solving eq 5, we can get optimal checkpoint interval T_c^{opt}. As an example, if $r=1$ from equation 5 we get,

$$T_c = \frac{2W(\frac{1}{2}\lambda n\sqrt{\frac{1}{\lambda n T_s}}T_s)}{\lambda n} \tag{6}$$

Where $W(z)$ is the Lambert W function.

V. Experiments and Results

A. Platform for experiments

Experiments were performed on a testbed of volunteer nodes managed by BOINC and Volpex. The testbed consisted of approximately 350 active volunteer nodes. Approximately 20% of the nodes were on the university campus and the remaining 80% were distributed worldwide. For the purpose of evaluation, Replica Exchange Molecular Dynamics (REMD) [6] code was executed repeatedly.

B. Measurement of parameters for checkpoint prediction

The key system parameter for estimating the optimal checkpoint interval is the failure distribution that is based on mean time to failure (MTTF) and the time to record a checkpoint. We recorded all failures of our system nodes for a substantial duration of time. We estimated the MTTF for the nodes in our volunteer pool using the following equation:

$$MTTF = \frac{\text{Total hours of operation}}{\text{Total number of failures}} \tag{7}$$

For the measurement made on our system, we get $MTTF = \frac{13678.67\text{ hrs}}{1714} = 28730$ second and $\lambda = \frac{1}{MTTF} = 0.0000348074$

Time to create checkpoint, T_s depends on various factors like checkpoint size, network congestion, link bandwidth, client/server configuration etc. We used the following cost function to calculate the T_s.

$$T_s = T_{s(cl)} + T_{s(lat)} + T_{s(up)} + T_{s(ack)} \tag{8}$$

where, $T_{s(cl)}$ = time to create checkpoint in client, $T_{s(lat)}$ = time to send checkpoint from client to server, $T_{s(up)}$ = time to upload checkpoint to server and $T_{s(ack)}$= time to send acknowledgment to client. The detailed procedure for estmating the checkpoint interval is discussed in [11].

Experiments were conducted with 50KB and 5MB checkpoint sizes. For 50KB checkpoints, the estimated checkpoint time was T_s = 1 second independent of the number of processes. For 5MB checkpoints, the estimated checkpoint time T_s = 156, 187 and 212 s for 16, 32 and 64 processes respectively. Finally, the predicted optimal checkpoint intervals were computed from the model developed in section IV using WolframAlpha's computational engines.

C. Experimental results

We executed the REMD code on 16 and 32 processes with no replicas, 2 replicas for each process and 3 replicas for each process. When going from 16 to 32 nodes, the problem size is scaled up (doubled) such that the computation workload per process is unchanged. The checkpoint sizes were 50KB and 5MB. A fixed checkpoint interval between 12 to 3200 seconds was employed for each run and the completion times were recorded. Each experiment was run ten times. Note that it is very important to have multiple runs for volunteer nodes as the execution time is expected to vary significantly between the runs based on the failure patterns of nodes in a run as well as the CPU and network characteristics of the specific sets of nodes selected for execution in each run.

Table I presents the application performance with predicted checkpoint intervals, along with the best and worst measured performance over the entire range of checkpoint intervals. Linear interpolation was employed for finding the job completion time for the predicted optimal checkpoint interval when the interval was between points for which measurements were made. We observe that there is often a significant difference between the predicted optimal checkpoint interval (T_c) and the checkpoint interval for which the lowest median execution time is recorded (T_c^{opt}). However, the actual execution time with the predicted optimal checkpoint interval is typically very close to the measured lowest median execution time. This is examined further in this section.

Checkpoint size, number of processes, degree of replication	Median of lowest completion time, T_{best} (s)	Median of highest completion time, T_{worst} (s)	Our predicted check-point interval, T_c (s)	Actual optimal check-point interval, T_c^{opt} (s)	Calculated completion time for T_c, $T_{predict}$ (s)	Abs % of difference between T_{best} and $T_{predict}$ (%)	Abs % of difference between T_{best} and T_{worst} (%)
50KB, 16, 1	3344	5191	42	50	3388	1.34	55.26
50KB, 16, 2	1946	2383	297	100	2213	13.74	22.43
50KB, 16, 3	1463	1973	851	50	1576	7.74	34.87
50KB, 32, 1	4362	6108	29	200	4398	0.82	40.03
50KB, 32, 2	2081	2388	235	25	2270	9.11	14.52
50KB, 32, 3	1348	1732	714	100	1671	23.93	28.49
5MB, 16, 1	5028	10333	465	800	6710	33.45	105.50
5MB, 16, 2	2217	4222	1708	3200	2604	17.50	90.46
5MB, 32, 1	10934	17298	339	1600	14331	31.08	58.21
5MB, 32, 2	2906	18374	1398	1600	3020	3.94	188.19
					Average	14.26	63.79
					Max	33.45	188.19

Table I: Performance comparison among measured best, worst and our predicted optimal checkpoint interval

We observe from Table I that percentage difference between the lowest measured completion time and completion time with the predicted optimal checkpoint interval (prediction error) ranges between 0.82% and 33.45% with an average prediction error of 14.26%. In a volunteer framework this is a low error given the inherent performance variations in a volunteer framework.

Table I also lists the percentage difference between the highest(worst) measured completion time and the completion time with the predicted optimal checkpoint interval, which ranges between 14.52% and 188.19% with an average difference of 63.79%. The point is that selecting a good checkpoint interval is important and selecting a poor checkpoint interval heuristically can have substantial implications for performance. The predicted checkpoint interval always leads to a performance fairly close to the optimal performance, and far superior than the performance with a poorly selected checkpoint interval.

Figure 1 shows the performance penalty with employing our predicted checkpoint interval, T_c (first set of bars) as well as employing fixed predefined checkpoint intervals for all scenarios in comparison with the measured lowest completion time in each scenario. Clearly using the predicted optimal checkpoint interval fares much better on average than employing any predefined checkpoint interval all the time. Moreover, even for the worst case, employing a predicted checkpoint yields a performance within 33% of the best measured performance, whereas it ranges between 39% and 188% for a fixed checkpoint interval.

Hence our predicted checkpoint interval is not truly optimal but it leads to an execution time close to the

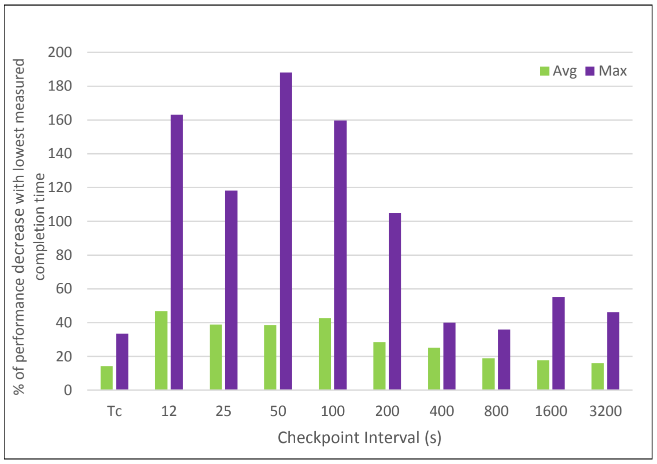

Figure 1: Performance penalty with always using a predefined fixed checkpoint interval versus the performance penalty using our predicted optimal checkpoint interval (T_c).

minimum possible job execution time. We did not know the true optimal checkpoint interval that results in the lowest completion time beforehand as it is determined empirically. The goal was to predict a checkpoint interval that yields an execution time close to the optimal and the results clearly demonstrate that the predicted checkpoint intervals are an excellent choice in practice. We do expect some error in identifying the optimal checkpoint interval given the nature of the volunteer environment where the execution performance typically varies significantly between runs. Also, this work makes a number of pragmatic assumptions detailed earlier (Section IV) that are the likely reasons for suboptimal experimental results.

VI. Concluding Remarks

This paper introduces a mathematically-based methodology to estimate a checkpoint interval that minimizes completion time for communicating inter-dependent parallel processes running in a high failure volunteer environment employing checkpointing and replication for fault tolerance. The results of the model are evaluated with the Volpex execution framework by running a real world application with various checkpoint sizes and replication. The results indicate that the model is effective in identifying a checkpoint interval that leads close to best possible application performance in this environment. Finally, while the checkpoint prediction model is evaluated in a volunteer environment, this work presents a fundamental contribution towards fault tolerant distributed systems. The results are applicable to other distributed systems such as computation clouds where replication and checkpointing are used together to gain acceptable performance in the face of failures.

VII. Acknowledgments

Partial support for this work was provided by the National Science Foundations Computer Systems Research program under Award No. CNS-0834750.

References

[1] Wolframalpha. http://www.wolframalpha.com.

[2] R. Anand, T. LeBlanc, E. Gabriel, and J. Subhlok, “A robust and efficient message passing library for volunteer computing environments,” *Journal of Grid Computing*, vol. 9, no. 3, pp. 325–344, 2011.

[3] D. P. Anderson, “BOINC: a system for public-resource computing and storage,” in *GRID ’04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 4–10.

[4] J. T. Daly, “A higher order estimate of the optimum checkpoint interval for restart dumps,” *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303–312, 2006.

[5] J. Daly, “A model for predicting the optimum checkpoint interval for restart dumps,” *Proceedings of the 2003 International Conference on Computational Science*, pp. 3–12, 2003.

[6] A. Dhar, A. Samiotakis, S. Ebbinghaus, L. Nienhaus, D. Homouz, M. Gruebele, and M. S. Cheung, “Structure, function and folding of phosphoglycerate kinase are strongly perturbed by macromolecular crowding,” *PNAS*, vol. 107, pp. 17 586–17 591, 2010.

[7] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. Scott, “An optimal checkpoint/restart model for a large scale high performance computing system,” in *IEEE International Symposium on Parallel and Distributed Processing*, April 2008, pp. 1–9.

[8] H. Nguyen, E. Pedamallu, J. Subhlok, E. Gabriel, Q. Wang, M. Cheung, and D. Anderson, “An execution environment for robust parallel computing on volunteer PC grids,” in *ICPP 2012: 2012 International Conference on Parallel Processing*, Pittsburgh, PA, Sep 2012.

[9] E. Pedamallu, H. Nguyen, N. Kanna, Q. Wang, J. Subhlok, E. Gabriel, M. Cheung, and D. Anderson, “A robust communication framework for parallel execution on volunteer PC grids,” in *CCGrid 2011: The 11th IEEE/ACM International Symposium on Clusters, Cloud and Grid Computing*, Newport Beach, CA, May 2011.

[10] S. Priya, M. Prakash, and K. Dhawan, “Fault tolerance-genetic algorithm for grid task scheduling using check point,” in *Sixth International Conference on Grid and Cooperative Computing, 2007*, Aug 2007, pp. 676–680.

[11] M. Rahman, H. Nguyen, J. Subhlok, and G. Pandurangan, “Checkpointing to minimize completion time for inter-dependent parallel processes on volunteer grids,” *coRR*, vol. abs/1603.03502, 2016. [Online]. Available: http://arxiv.org/abs/1603.03502

[12] S. Ross, *Applied Probability Models with Optimization Applications*. Dover Publications, 1992.

[13] J. W. Young, “A first order approximation to the optimum checkpoint interval,” *Communications of the ACM*, vol. 17, no. 9, pp. 530–531, 1974.