Monetary Cost Optimizations for HPC Applications on Amazon Clouds: Checkpoints and Replicated Execution

Yifan Gong
NEWRI
Interdisciplinary Graduate School
Nanyang Technological University, Singapore

Amelie Chi Zhou
School of Computer Engineering
Nanyang Technological University
Singapore

Bingsheng He
School of Computer Engineering
Nanyang Technological University
Singapore

I. Motivation

Recently, we have witnessed that many emerging high performance computing (HPC) or scientific computing applications are developed and hosted in the cloud. As those applications are usually long running jobs and are costly in the cloud, monetary cost [11], [7] and performance [3], [2] are important optimization factors. Message Passing Interface (MPI) is the key programming paradigm for developing HPC and scientific applications. That motivates us to investigate whether and how we can reduce the monetary cost for MPI-based applications with performance constraint in the cloud.

Cloud has evolved into an economic market. Besides *on-demand* instances that charges users at a fixed rate, Amazon EC2 provides spot instances, whose prices are mainly determined by the supply and demand in the market. Table I shows the statistics of the price history of four types of spot and on demand instances on Amazon in the US East region during August 2013. We have the following observations: a) Spot instances are usually much cheaper than on-demand instances. There are some "outlier" points where the maximum price is much higher than the on-demand price. If spot instances are leveraged properly, they can reduce monetary cost [10], in comparison with the solutions with on-demand only. b) Different instance types have different variations on the price. These observations are consistent with the previous studies [6].

Leveraging spot instance is an ideal approach to reduce the monetary cost of MPI executions. However, a spot instance can be terminated whenever the spot price is higher than the bidding price (i.e., an out-of-bid event). We have observed that the spot price is highly dynamic in both spatial and temporal dimensions. For spatial dynamics, clouds (e.g., different Amazon EC2 zones) have very different spot prices. For temporal dynamics, spot prices can be rather stable for some times, and be changing dramatically for other times. Due to the spot price dynamics, failures can occur in MPI executions. In order to satisfy the performance requirement (usually in the form of deadlines), fault tolerant executions are necessary. In this paper, we investigate two common fault-tolerant mechanisms of MPI, including checkpointing and replicated execution. These two mechanisms are actually complementary with each other. Checkpointing can reduce the execution time when the failure occurs and replicated execution can reduce the failure rate in spot market. When the spot price is stable, checkpointing is not necessary. When the spot price varies sharply, checkpointing technique becomes more useful.

TABLE I. Statistics on spot prices ($/hour, August 2013, US East Region) and on-demand prices of Amazon EC2.

Instance type	Average	$stdev$	Min	Max	OnDemand
m1.small	0.048	0.438	0.007	10	0.06
m1.medium	0.246	1.31	0.0001	10	0.12
m1.large	0.069	0.770	0.026	40	0.24
m1.xlarge	0.413	2.22	0.052	20	0.48

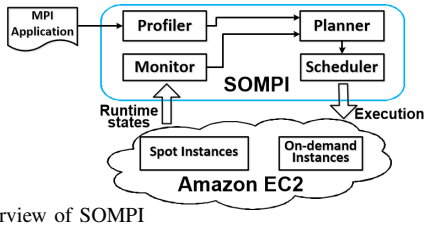


Fig. 1. Overview of SOMPI

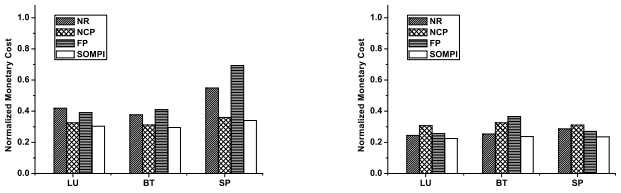
In this paper, we propose a runtime system called SOMPI that minimizes the monetary cost on Amazon EC2 clouds and has the same interfaces as MPI. We execute MPI-based applications with the hybrid of both spot and on-demand instances from Amazon EC2. The runtime dynamically decides the checkpointing and replication settings. We propose a cost model to guide the design of bidding price and the combination of checkpoint and replication techniques to have the most cost-effective reliable executions. We perform experimental studies with NPB benchmarks on Amazon EC2 and achieve up to 65% monetary cost reduction over using on-demand instances only while satisfying the deadline requirement.

II. System Overview

We design SOMPI runtime system to minimize the monetary cost for MPI-based applications while satisfying users' performance requirements. Figure 1 gives an overview on the structure of SOMPI. SOMPI has four key components: profiler, planer, scheduler and monitor. We present the functionality of each component one by one.

Profiler. The profiler analyzes the runtime performance of applications and provides the profiling results to the planner to make optimization plans. The results include the communication and computation performances of the applications, such as the instruction-level statistics, memory usage and communication patterns. There are a number of ways to profile parallel programs [8] and we adopt TAU (Tuning and Analysis Utilities) [9] as the profiling tool in our implementation.

Planner. The planner takes the profiling results as input to build a cost model. The cost model makes the important optimization decisions, including the instance configurations



(a) Stable Trace (b) Unstable Trace

Fig. 2. The overall comparison for different approaches under the same deadline constraint (64 medium instances, with one MPI process on each)

for MPI applications, bidding prices for spot instances and the combination of the two fault-tolerant mechanisms (i.e., checkpointing and replication).

Scheduler. The scheduler is designed to deploy applications on Amazon EC2 according to the optimization decisions generated by planner. Specifically, scheduler is responsible for aquiring/releasing the spot/on-demand instances as needed, assigning applications to preferred type of instances and restarting processes in case of instance failure.

Monitor. During the application execution, the monitor traces the runtime states of the applications and the cloud (spot price variance), which are later sent to the planner to refine the optimization decisions at runtime.

III. Technique Overview

The core of SOMPI is that we develop a holistic approach of combining the two classic fault-tolerant mechanisms: checkpointing and replicated execution. They are complementary to handle different spot dynamics. Moreover, we are facing a series of complicated issues on their interplays with the spot instances. We further develop a cost model to automatically determine the design of bidding price and the combination for these two fault-tolerant mechanisms. We present more details of our system in the technical report [4].

Checkpointing. We implement our runtime system based on an system-level coordinated non-locking checkpointing strategy. It does not require source-level modifications to significantly pose runtime overhead of the MPI application. To make checkpointing cost-effective, we consider the design issues in different bidding price and spot trace.

Replicated Execution. We use active replications to implement our system. The spot prices in different replications are independent. To make replicated executions cost-effective, we consider the design issues including when to use replicated execution and how to set the bidding price.

Cost model. The cost model involves many constraints and optimization derivations. We formulate the problem as a nonlinear optimization problem with nonlinear constraints. We use the method of Lagrange multipliers [5] for finding the local minimal solution. With such a formulation, we can find the cost-effective solution in a timely manner.

IV. Evaluations

We apply our runtime system to NAS Parallel Benchmarks (NPB) [1] kernels version 2.4. The benchmarks have three applications, including BT (Block Tri-diagonal solver), SP (Scalar Penta-diagonal solver) and LU (Lower-upper Gauss-Seidel solver). In order to demonstrate the effectiveness of each proposed optimization component, we developed four variants of SOMPI, including Baseline, FP, NR and NCP. Baseline only adopts on-demand instances to run MPI-based applications. FP fixes the bid price of the spot instances as the on-demand price of the same type and calculates the optimal checkpointing frequency. NR does not use the replication and NCK executes the applications without checkpoint.

Figure 2 shows the monetary cost normalized with the cost of Baseline. The deadline is set to be 20% larger than the execution time of the Baseline. We have two major observations. First, our SOMPI can largely reduce the monetary cost compared with other approaches. It can reduce up to 65% monetary costs than using on-demand instances only. It indicates the importance of leveraging spot instances. Second, SOMPI is only about 5% better than NCP when the trace is stable. It indicates that checkpointing techniques plays a minor role on the stable spot trace. When the trace varies sharply, monetary cost of SOMPI and NR are similar, and checkpointing plays a more important role.

V. Conclusion

Amazon EC2 spot instances give us a chance to reduce monetary costs of HPC applications compared with on-demand instances only. In our work, we propose a runtime system called SOMPI for MPI applications and leverage both spot and on-demand instances for monetary cost optimizations. We developed a cost model guided approach to combine checkpoint and replication. We implement SOMPI on Amazon EC2 and evaluate its efficiency with NPB benchmarks. Our experimental results show that SOMPI can reduce up to 65% monetary costs than using on-demand instances only while satifying the deadline requirement. More details about this project can be found in our technical report [4].

VI. Acknowledgments

We acknowledge the support from the Singapore National Research Foundation under its Environmental & Water Technologies Strategic Research Programme and administered by the Environment & Water Industry Programme Office (EWI) of the PUB, under project 1002-IRIS-09. Yifan is supported by the scholarship from NEWRI, IGS (Interdisciplinary Graduate School). Bingsheng is in part supported by a MoE AcRF Tier 1 (2014-T1-001-145) of Singapore.

References

[1] E. Barszcz, J. Barton, L. Dagum, et al. The nas parallel benchmarks. In *Supercomputer Applications*, 1991.

[2] Y. Gong, B. He, and D. Li. Finding constant from change: Revisiting network performance aware optimizations on iaas clouds. In *SC*, 2014.

[3] Y. Gong, B. He, and J. Zhong. Network performance aware MPI collective communication operations in the cloud. *TPDS*, 99:1, 2013.

[4] Y. Gong, A. C. Zhou, and B. He. Monetary cost optimizations for hpc applications on amazon clouds: Checkpoints and replicated execution. Technical Report 2014-TR-120, NTU, http://pdcc.ntu.edu.sg/xtra/tr/2014-TR-120-SOMPI.pdf, 2014.

[5] J. L. Lagrange. *Mécanique analytique*, volume 1. Mallet-Bachelier, 1853.

[6] W. Lu, J. Jackson, and R. Barga. Azureblast: a case study of developing science applications on the cloud. In *HPDC'10*, pages 413–420, 2010.

[7] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *SC*, 2012.

[8] S. Moore, D. Cronk, K. S. London, and J. Dongarra. Review of Performance Analysis Tools for MPI Parallel Programs. In *PVM/MPI*, pages 241–248, 2001.

[9] S. S. Shende and A. D. Malony. The tau parallel performance system. *IJHPCA*, 20(2):287–311, 2006.

[10] S. Yi, A. Andrzejak, and D. Kondo. Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *IEEE TSC*, 5(4):512–524, 2012.

[11] A. C. Zhou and B. He. Transformation-based monetary cost optimizations for workflows in the cloud. *TCC*, pages 1–1, 2014.