Process Replication for HPC Applications on the Cloud

Scott Purdy and Pete Hunt
Advised by Prof. David Bindel

December 17, 2010

1 Abstract

Cloud computing has emerged as a new paradigm in large-scale computing. High-performance computing (HPC) is an important research tool for a variety of fields. HPC requires a large investment in specialized computing infrastructure. While these costs could be alleviated by commercial cloud computing environments, standard clouds have extremely variable latency and reliability characteristics and are unsuitable for traditional HPC applications. In this paper we explore an approach to alleviate the latency limitations of cloud environments for HPC applications.

2 Introduction

Traditional HPC applications perform poorly in cloud computing environments because of the latency and, sometimes, bandwidth characteristics of the cloud. On Amazon EC2, currently one of the most widely used cloud computing environments [4], the point-to-point latency between all instances (virtualized HPC nodes) is extremely variable. Due to this variability, many classes of HPC applications, such as bulk-synchronous parallel (BSP) applications, perform poorly because they are bound by the highest observed latency per iteration. This is unfortunate, as cloud computing is touted to be the platform of the future and represents attractive economics for HPC researchers.

Additionally, the failure rate in a virtualized cloud environment is higher than that of a traditional, dedicated HPC cluster [1]. In many HPC applications a single node failure will require a restart of the entire computation since the last checkpoint.

We propose a new model called process replication that can be used to improve performance and fault-tolerance of HPC applications on the cloud. HPC applications involve a set of processes coordinating across a cluster of machines using message passing. The defacto standard for message passing is known as the Message Passing Interface, or MPI. We propose using state-machine replication [9] to create several replicas of each process across different instances in the cloud.

Because each replica sends and receives the same set of messages, the latency of the first message an individual receives from the set of replicas will be lower on average than without replication. For messages sent simultaneously from each replica in a set, the expected time it takes for the first message to be received by a specific recipient reduces as the replication fac-

tor increases. We show this empirically below.

Process replication is not without its drawbacks. It requires that individual processes are deterministic and can be replicated without introducing conflicting states. It also trades bandwidth, CPU time and memory for latency. We view this as an attractive trade-off for latency bound applications. It is also becoming increasingly advantageous because CPU, memory and bandwidth costs and performance are improving much faster than latency [8].

3 Previous Work

With the rise of commercial cloud environments such as EC2, the scientific community has seen the potential for low-cost, easy-to-use elastic cluster computing applications. As such, there have been several studies of the characteristics of EC2 with a specific eye to reliability and performance.

Evangelinos and Hill studied HPC applications running on EC2 [3]. They found a wide range of latency and bandwidth performance between various MPI implementations. The MPI performance on EC2 was, however, inferior to that of a dedicated cluster. Additionally, they found the memory and CPU usage of EC2 instances to be high. The overall conclusion of the study was that performance was comparable to a low-cost cluster configuration but was below that of dedicated supercomputing centers. They also proposed the creation of a dedicated HPC cloud infrastructure which Amazon rolled out two years later [7].

Hazelhurt corroborated these findings, and found that EC2 instances performed quite well with single-threaded tasks on a single node but performance degraded as the number of threads increased[5]. Additionally, he found that after 63 instances the efficiency on his benchmark would start to decrease. EC2 actually outperformed one dedicated cluster on the benchmark when the number of nodes grew beyond 12. He also did a cost comparison and found that it is cheaper to build a network of workstations only when there is greater than 10-50% utilization over a machine lifetime of 3 years.

Ke, Burtscher and Speight explored the idea of runtime MPI compression on traditional HPC clusters and saw performance gains up to 98% [6]. Most benchmarks saw improvements between 3% and 27% and none performed worse. They observed that tasks with a high communication to computation ratio saw the most improvement and became increasingly limited by the rate of compression. The compression overhead for most benchmarks was under 2%; the highest was under 6%. Because EC2's characteristics tend to increase the cost of communication MPI compression is a promising area of exploration for cloud computing.

Amazon EC2 is powered by the Xen virtualization infrastructure. Youseff, Wolski, Gorda and Krintz [10] found that while Xen had a perceived overhead, it imposed no statistically significant overhead for scientific computation and actually exhibited improved latency performance due to some specifics of the Xen I/O implementation.

4 Properties of MPI on Amazon EC2

We conducted a series of experiments and benchmarks to obtain a distribution of point-to-point latency between instances on Amazon EC2. We used the "us-east-1b" availability zone (cho-

sen at random) and used the "m1.small" instance type. While higher-performance instance types are available, they were unnecessary for this benchmark as we were merely measuring network latency, not CPU performance. Additionally, we opted not to use the Amazon cluster compute ("cc1.4xlarge") instance type as we are only interested in performance on general-purpose cloud environments, not specialized HPC ones.

We wrote a benchmark in C utilizing OpenMPI on a set of 20 instances running Ubuntu Linux. This benchmark had each node send ping/pong messages to every other node in the cluster in a round robin fashion. The results were communicated to the master node in CSV format. The aggregated CSV file containing (sender, receiver, latency) triples was analyzed to create the simulated EC2 distribution depicted in Figure 1. It is apparent that the latency has a strong positive skew and a high variance. The distribution is summarized in Figure 2.

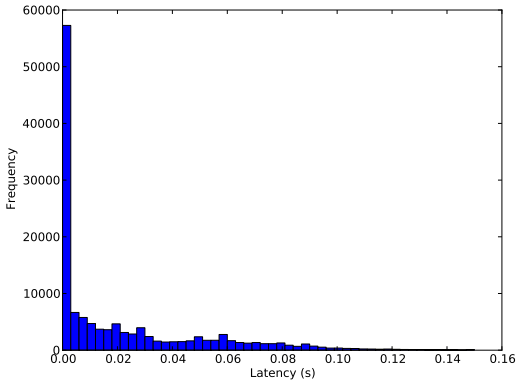

Figure 1: The latency distribution sampled from a round robin ping/pong benchmark running on twenty Amazon EC2 instances. This figure is zoomed in on the range [0, 0.16] and there is a long tail that extends out to 0.47.

5 Process Replication

The core idea of process replication is to perform each unit of computation at multiple nodes in a cluster. This is in contrast to performing each unit of computation a single time at a single node in a cluster. The nodes used for process replication can be new nodes or existing nodes. Each of the nodes that performs a given computation sends results to other nodes that need them. The first results received are used, since they should be identical to any future results for a given unit of computation (we leave the encapsulation of non-determinism to future work).

There are two specific advantages of this model. First, since the variance of network la-

Replication factor	1	2
Mean (s)	0.0217	0.0086
Standard deviation (s)	0.0295	0.0172
Minimum (s)	0.0002	0.0002
Maximum (s)	0.4722	0.0993

Figure 2: Summary of the Amazon EC2 latency distribution at two levels of replication. The replication factor of two is computed using the minimum of two latency samples for each iteration, which is a close, lower-bound approximation of the actual latency impact.

tency is large in the cloud environment, the latency of the first results received has a much lower expected value than the latency of any given message. Figure 4 shows the expected time for a latency-bound computation task with different replication factors.

The second advantage is fault tolerance. Without using process replication, the entire computation must be restarted from the last checkpoint in the event of a process failure. This increases the computation time proportionally with mean time to failure and also results in increased administrative work. Additionally, many applications do not implement checkpointing and must be restarted from the beginning. With process replication, the system becomes much more resilient; the computation can continue as long as at least one replica of every process has not failed. That is, in an N-process computation with a replication factor of R, the system can survive $R-1$ failures per process. This means the entire system can survive a minimum of $R-1$ failures and a maximum of $N(R-1)$ failures. Without process replication, $R=1$ and the system cannot survive a single failure.

Process replication reduces the latency of each individual point-to-point message. This means that it can be useful for almost any distributed computation. However, there are some applications in which process replication can provide substantially larger latency improvements. These include bulk synchronous and iterative problems, such as the conjugate gradient method, which we use as our benchmark below. In general, computations that are latency-bound can benefit greatly while CPU- or bandwidth-bound computations will still realize improvements in fault-tolerance but may see little or negative performance effects.

One of the compelling aspects of the process replication model is that it is not necessary to increase the number of discrete nodes in the system. Instead, existing nodes can be used as replicas for each other. Since our work is focused on latency-bound problems, this set up does not slow down the computation significantly compared with the speedups in latency. And because some nodes will contain replicas for two communicating processes, the network communication cost can be eliminated in some cases. For the purposes of isolating the latency improvements to actual network communications, our benchmarks use new nodes as replicas.

6 Experimental Methodology

We generated a distribution of the observed latency as described above. Additionally, we designed our benchmarks to emit a CSV file that logged all communication patterns of the benchmark. This CSV file, along with the latency distribution, were processed by a simulation program to compute the computation time for varying replication factors. This simulator utilized a sample of 131418 latency observations from a twenty instance cluster of EC2 small instances as discussed above. For every time step in the computation log, the simulator computed the estimated network latency by drawing R, the replication factor, samples from the observed distribution and taking the minimum of these observations. The estimated latencies for each time step were summed and the result given as the total computation time. Note that this model makes the assumption that CPU time is negligible and the problem is strictly latency-bound.

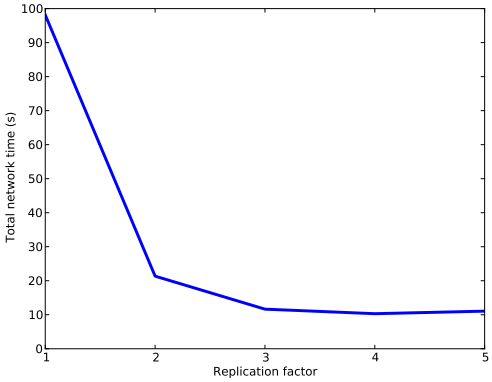

Figure 3: Total running time for an 8-node mock BSP computation at varying levels of replication on EC2.

7 Experiment 1: Mock BSP Benchmark

We began our experiments by devising a very simple benchmark to simulate the general case of a bulk-synchronous-parallel problem. The benchmark ran four processes on EC2 which all communicated with each other before moving to the next time step. A graph of these performance statistics over 10000 iterations can be found in Figure 3. For each of the four processes, we launch R nodes, where R is the replication factor. Each of the R nodes for a given process sends the same set of messages and each node in the system uses the first message from a replication set immediately. Duplicate messages from the other replicas that are received later are ignored. This benchmark shows that for the truly latency-bound scenario there can be significant reductions in the effective latency.

8 Experiment 2: NAS CG Benchmark

In order to evaluate the effects of process replication on a real-world application, we have instrumented the NAS CG benchmark to log communication patterns throughout the computation. The NAS CG benchmark was chosen because the NAS Parallel Benchmarks are a standard suite of MPI benchmarks and we wanted to see how process replication would affect an existing parallel application with minimal modification. Additionally, the conjugate gradient (CG) benchmark is a latency-bound bulk-synchronous parallel computation which would theoretically benefit significantly from process replication.

The benchmark is written in Fortran and was modified to log all communication by wrapping the OpenMPI implementation with logging calls. No other modifications were made. The log is used for simulating process replication results as described in the next section.

The benchmark iteratively computes the standard conjugate gradient method over 8 instances. We ran this benchmark on a single machine to generate the logs as the simulator cares only about communication patterns and not about the actual runtime performance of the environment the logs were generated on. We then took this log and the observed latency distribution and ran the simulator to generate our results. As shown in Figure 4, our results for this real-world benchmark are quite similar to those in our mock BSP benchmark.

9 Future Work

The primary remaining task is to develop an industrial-strength implementation that exploits

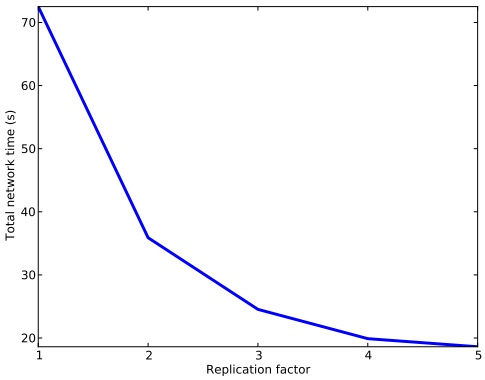

Figure 4: Total simulated network latency for an 8-node NAS CG benchmark at varying levels of replication.

these ideas. The specific challenges lie in a more complete MPI implementation and encapsulating nondeterminism such as synchronized clocks, random number generation and message ordering. With respect to the latter point, there has been promising research in implementing state-machine replication within the hypervisor [2], which is attractive in virtualized environments and would minimize modifications to existing applications. Additionally, compression seems to be "almost free" according to Ke, Burtscher and Speight and could mitigate the latency-bandwidth trade-off resulting from using the process replication method.

10 Conclusion

Process replication is a viable solution for many classes of HPC applications running on the cloud. Increasing the replication factor provides a way to trade bandwidth, CPU time and memory for improved latency, a trade-off that was previously difficult to achieve or otherwise impossible. And because latency is very expensive and the limiting factor for many applications, this potentially order-of-magnitude improvement is a very attractive solution. Additionally, process replication brings the benefit of fault-tolerance, which is desired in such dynamic, changing and large-scale environments like commercial cloud platforms. Overall, process replication will help to make cheap, high-performance cloud computing a viable platform for HPC applications.

References

[1] Daniel J. Abadi. Data management in the cloud: Limitations and opportunities. *Bulletin of the Technical Committee on Data Engineering*, 32:3–12, March 2009.

[2] Thomas C. Bressoud and Fred B. Schneider. Hypervisor-based fault tolerance. *ACM Trans. Comput. Syst.*, 14:80–107, February 1996.

[3] Constantinos Evangelinos and Chris N. Hill. Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2. 2008.

[4] Jeremy Geelan. The top 150 players in cloud computing. October 2009.

[5] Scott Hazelhurst. Scientific computing using virtual high-performance computing: a case study using the Amazon Elastic Computing Cloud.

[6] Jian Ke, Martin Burtscher, and Evan Speight. Runtime compression of MPI messages to improve the performance and scalability of parallel applications. 2004.

[7] Jeremy Kirk. Amazon introduces cluster computing for HPC apps. July 2010.

[8] David Patterson. Why latency lags bandwidth, and what it means to computing. MIT Lincoln Labs, October 2004.

[9] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22:299–319, December 1990.

[10] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Evaluating the performance impact of Xen on MPI and process execution for HPC systems. 2006.