Exploring the Performance Fluctuations of HPC Workloads on Clouds

Yaakoub El-Khamra[3,4], Hyunjoo Kim[1], Shantenu Jha[2], Manish Parashar[1]

[1] *NSF Center for Autonomic Computing, Dept. of Electrical & Computer Engr., Rutgers University, , NJ, USA*

[2] *Center for Computation & Tech. and Dept. of Computer Science, Louisiana State University, USA*

[3] *Texas Advanced Computing Center, The University of Texas at Austin, Austin Texas, USA*

[4] *Craft and Hawkins Dept. of Petroleum Engineering, Louisiana State University, USA*

Abstract

Clouds enable novel execution modes often supported by advanced capabilities such as autonomic schedulers. These capabilities are predicated upon an accurate estimation and calculation of runtimes on a given infrastructure. Using a well understood high-performance computing workload, we find strong fluctuations from the mean performance on EC2 and Eucalyptus-based cloud systems. Our analysis eliminates variations in IO and computational times as possible causes; we find that variations in communication times account for the bulk of the experiment-to-experiment fluctuations of the performance.

Categories and Subject Descriptors D.1.3 [Concurrent Programming]: Distributed Programming, Parallel Programming **General Terms** Algorithms, Performance, Experimentation **Keywords** Cloud Computing

I. Introduction and Motivation

Emerging cloud platforms present relatively simple computing environments with attractive service-oriented abstractions of resource management, capacity planning capabilities, software deployment & control, etc. Clouds are emerging as an important class of distributed computational resources and are quickly becoming an integral part of production computational infrastructures to support high-performance computing (HPC) workloads. In spite of the promise of clouds as a viable platform for science & engineering applications, many fundamental questions persist about how scientific applications will utilize clouds, both now and into the future?

A key challenge is the large fluctuations in performance experienced by HPC workloads on clouds, which can significantly impact the effectiveness of this platform for many computational applications. For example, when using clouds as accelerators or to respond to spikes in resource demand, an implicit assumption in deadline-driven schedulers and/or smart schedulers is that the runtime can be well estimated. Whereas much effort has been spent on modeling effective and accurate runtime estimation techniques for a given kernel [1], [2], little attention has been spent on estimating inherent *fluctuations* in their runtime due to the cloud operational environment. The typical experiment-to-experiment fluctuations for any given infrastructure configuration can be an important component of the overall time-to-solution; thus it is critical to measure and understand these fluctuations.

In many ways, the fluctuations in the time-to-solution are analogous to the fluctuation in *queue wait-time* for regular batch-queue schedulers, in that it can be dependent on system load, and related factors. In other ways, the fluctuation appears to be independent of the system load and a consequence of the virtualization layer and related system properties. For traditional shared HPC/clusters under normal conditions, the fluctuations in runtime are typically small compared to queue wait-times, hence traditionally ignored. This paper is an attempt to explore and characterize runtime fluctuations for a given application kernel – which is representative of both a large number of MPI/parallel workloads and workflows, on two different cloud offerings – Amazon's EC2 and Eucalyptus (as provided by NSF's FutureGrid (FG) [3]). As it is difficult to discern or reverse engineer the specific infrastructure details on EC2, we use a controlled and well understood environment (FG see §III) in order to correlate the fluctuations to infrastructure details.

II. Related and Prior Work

In Ref. [4] we considered an autonomic approach to integrated HPC grid and cloud usage, which we extended to explore application and infrastructure adaptation on such hybrid infrastructure [5]. We established that clouds in conjunction with traditional HPC and high throughput computing (HTC) grids provide a balanced infrastructure supporting scale-out and scale-up/down for a range of application sizes and requirements. We demonstrated that

978-0-7695-4302-4/10 $26.00 © 2010 IEEE
DOI 10.1109/CloudCom.2010.84

System Type	Cores	TFLOPS	RAM (GB)	Site
IBM iDataPlex	1024	11	3072	IU
Dell PowerEdge	768	8	1152	TACC
IBM iDataPlex	672	7	2016	UC
IBM iDataPlex	672	7	2688	SDSC
Total	3136	33	8928	

TABLE I: Compute hardware for FutureGrid: Dynamically configurable systems

System Type	Cores	TFLOPS	RAM (GB)	Site
Cray XT5m	672	6	1344	IU
TBD System	480	6	640	IU
IBM iDataPlex	256	2	768	UF
High Throughput Cluster	384	4	192	PU
Total	1792	16	2944	

TABLE II: Compute hardware for FutureGrid: Systems not dynamically configurable

such hybrid infrastructure supports a range of objectives, for example, reduced times-to-solutions, reduced costs in terms of currency or resource usage, or resilience against unexpected outages (e.g., unexpected delays in scheduling queues or unexpected failures). We also investigated how clouds can be effectively used as accelerators to address changing computational requirements as well as changing Quality of Service (QoS) constraints (e.g., deadlines) for a dynamic application workflow. Furthermore, we explored how application and system level adaptivity can be used to achieve improved overall application performance and more effective utilization of the hybrid platform.

In Ref. [6] Wang et. al. characterized the impact of virtualization on the networking performance of the Amazon EC2. They measured the processor sharing, packet delay, TCP/UDP throughput and packet loss among EC2 virtual machines and showed that even though the network is lightly utilized, virtualization can cause significant throughput instability and abnormal delay variations. Thus, there can be performance variation not only for different infrastructures and application configurations, but also for different trials of a given experiment as resources such as processors, network drivers, etc. are shared between users.

III. An Overview of FutureGrid

A significant fraction of our experiments and analysis are performed on the FG [3] – the US NSF's Experimental Grid testbed to allow scientists to collaboratively develop and test innovative approaches to parallel, grid, and cloud computing. The ultimate goal is to build a system that helps researchers identify cyberinfrastructure that best suits their scientific needs. The FG testbed will be composed of a high-speed network connected to distributed clusters of high performance computers (India, Sierra, Alamo, X-ray) and will be linked to the TeraGrid. FG is a (small 5600 core) Science Cloud and employs virtualization technology that allows the test bed to support a wide range of operating systems. It opened for early users in June 2010.

hines, specifications and configurations of computing hardware are described in tables I and II. The individual nodes contain dual socket quad core Nehalem X5590 processors providing a total of 8 physical cores. Eucalyptus is configured to provide a maximum of 8 virtual cores mapped directly to the physical cores (HyperThreading is turned off) and the host system is run on core 0.

IV. Workload Description

In this investigation we attempt to reproduce the results of our earlier investigations on EC2 using the Kalman Filter-based history matching workflow [5]. The workflow consists of an ensemble of reservoir simulation jobs that run for a defined period of simulation time and perform a collective analysis step at regular intervals. The whole workflow has several "stages" and the simulation and analysis steps are repeated in each stage. The analysis step acts as a collective barrier: all jobs needs to reach the analysis step before simulations can proceed to the next stage.

The reservoir simulations can use a block Jacobi preconditioner with three solvers: Generalized Minimal Residual Method (GMRES), Conjugate Gradient (CG) or Bi-Conjugate Gradient (BiCG). In general, GMRES methods perform a large number of reductions, which translates to a large number of MPI_Allreduce function calls compared to CG and BiCG. Therefore, it involves more collective MPI operations than CG or BiCG. In comparison, CG and BiCG are faster methods involving one matrix vector multiplication step per iteration but do not monotonically decrease the residual and do not guarantee convergence.

The reservoir simulations can run on 1,2,4 or 8 cores of a c1.xlarge instance. To utilize multiple cores, we ran the reservoir simulations using MPI. Furthermore, we have the option of running any of three problem sizes. Therefore, we have a total of 72 benchmarks (4 core counts, six solver combinations, three problem sizes). We run each benchmark 12 times to obtain sufficient statistical sampling.

It is worth mentioning that FG machines are in early/friendly user mode and while the hardware is final, a significant portion of the software stack is still being deployed and configured on machines *India*, *Sierra* and *Tango*. The image we used is the same image we have used in previous work [5] and is publicly available on both India (ami-FF290BF5) and EC2(ami-8df619e4). The image contains gcc 4.1.2 compilers which we used to build mpich2-1.2 and PETSc-3.1 [7] with full optimization.

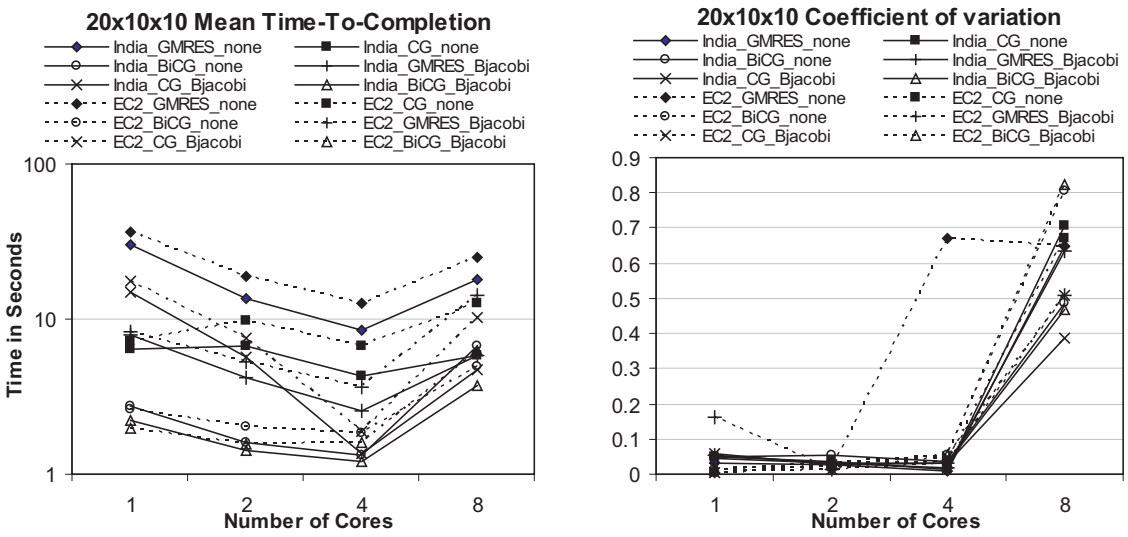

Fig. 1: Benchmark results for simulations of size 20x10x10 with different solvers (GMRES, CG, BiCG), with and without a block Jacobi preconditioner. The legend should be read from left to right and then top to bottom. Solid lines correspond to India benchmark results and dotted lines corresponding to EC2 benchmark results. Benchmarks ran on c1.xlarge instances on India and EC2 with MPI. The plot on the left shows the average time to completion for the 12 runs made and the plot on the right shows the coefficient of variation.

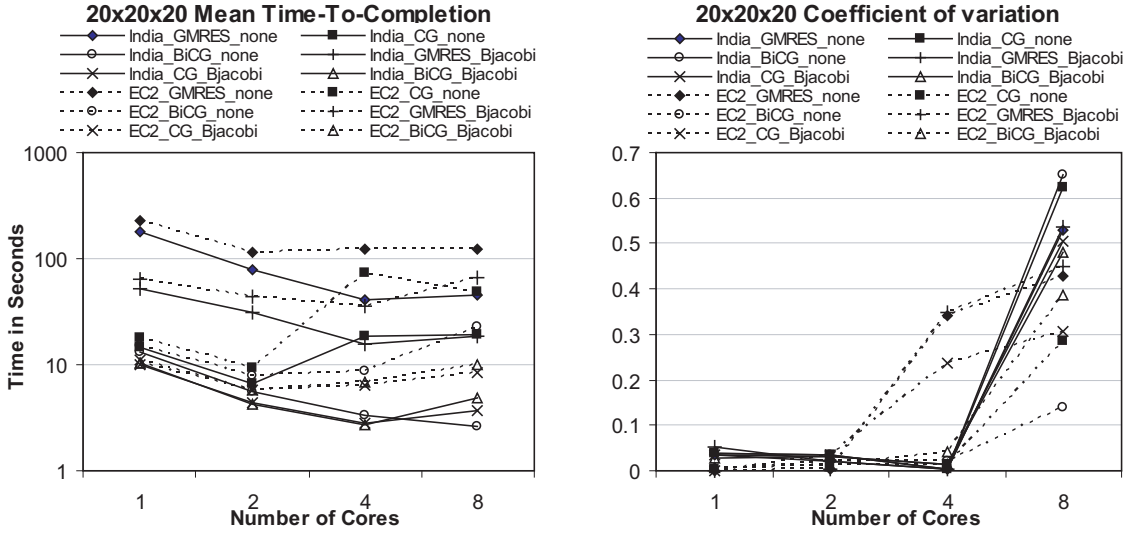

Fig. 2: Benchmark results for simulations of size 20x20x20 with different solvers (GMRES, CG, BiCG), with and without a block-Jacobi preconditioner. The legend should be read from left to right and then top to bottom. Solid lines correspond to India benchmark results and dotted lines corresponding to EC2 benchmark results. Benchmarks ran on c1.xlarge instances on India and EC2 with MPI. The plot on the left shows the average time to completion for the 12 runs made and the plot on the right shows the coefficient of variation.

V. FutureGrid Experiments and Analysis

Consistent with our earlier work, we experiment with six application configurations, on two different cloud infrastructures (EC2, FG/Eucalyptus). In order to understand the inherent fluctuations in the runtime due to the cloud operational environment (See § I), and determine the contribution of IO to these fluctuations, we ran two sets of benchmarks – with and without IO, on suspicion that IO would impact performance. The results did not vary greatly between the two sets; the difference in the coefficient of variation (defined as the ratio of standard deviation and the mean) for each of the benchmarks was on average less than 1%. Furthermore, the IPM [8] profile indicated that IO is not

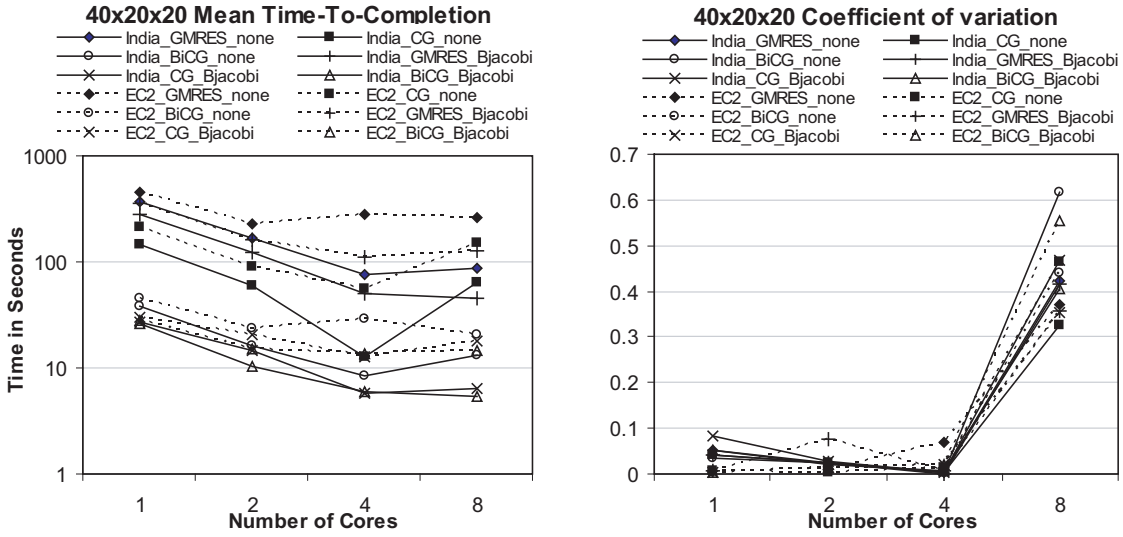

Fig. 3: Benchmark results for simulations of size 40x20x20 with different solvers (GMRES, CG, BiCG), with and without a block-Jacobi preconditioner. The legend should be read from left to right and then top to bottom. Solid lines correspond to India benchmark results and dotted lines corresponding to EC2 benchmark results. Benchmarks ran on c1.xlarge instances on India and EC2 with MPI. The plot on the left shows the average time to completion for the 12 runs made and the plot on the right shows the coefficient of variation.

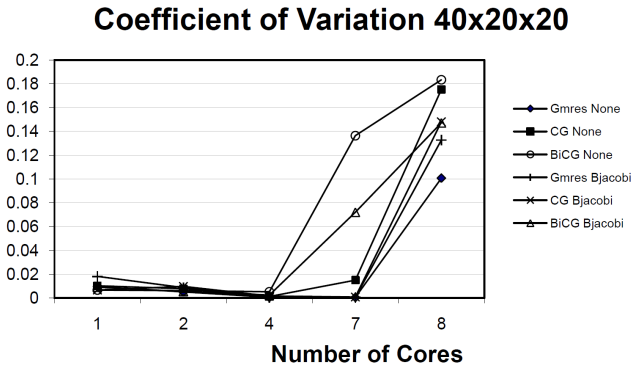

Fig. 4: The coefficient of variation for the six solver/preconditioner combinations on a 40x20x20 problem run on India c1.xlarge without IO. Notice the relatively large coefficient of variation at 7 cores for some of the benchmarks.

a bottleneck. Therefore, we will be restricting our discussion to a typical, real-world, IO enabled benchmark.

Results are presented in Figures 1-3. The plots on the left are the average run time in seconds for 1,2,4 and 8 cores for each of the six combinations of solver/preconditioner for three problem sizes. The plots on the right are the coefficient of variation for the benchmarks.

It is important to note that only some of the benchmarks exhibit a sharp increase in performance variation starting at 8 cores (Figures 1-3). A large number of benchmarks exhibit this behavior at 7 cores (Figure 4). The performance variation therefore is not restricted to cases where the benchmarks are run on all cores of a VM and host. This asserts our belief that the host OS load is minimal and is not the primary cause of the performance variation.

Inspecting our results, it is obvious that for each problem size there exists an optimal solver/preconditioner combination for a particular number of cores. This is the case for both EC2 and FG. Furthermore, there is some similarity in the performance trends for EC2 and FG. It is also obvious that FutureGrid is a considerably faster platform, due partly to the hardware infrastructure: FutureGrid has 2.93Ghz Nehalem processors while EC2 has 2.33 Ghz Quad Core Xeon processors.

In terms of computational cost, most of the benchmarks complete on a single core in under three minutes, with the exception of GMRES without a preconditioner on the largest problem size, which takes 8 minutes on EC2 and 6 minutes on India. The total Flop count (sum of Flops from all cores) ranges from $3.463e+10$ to $2.560e+11$ and is constant for each benchmark, i.e. a benchmark performs the same number of flops every time. The Flops/sec varies considerably from one repetition of the benchmark to another.

The variation is attributed, at least in part, to a large variation in the time spent in communication. For example on India under normal conditions, GMRES with no preconditioner and with the largest problem set requires 9.8 seconds and spends 31.54% of this time performing communication. The MPI function MPI_Allreduce consumed 24.96% of the wall clock time and MPI_Waitany consumed 3.29% of the wall clock time. In a subsequent run of

the same benchmark in the same instance, the benchmark consumed 29.5 seconds and spent 78.079% of the time in communication. MPI_Allreduce consumed 65.41% of the wall clock time and MPI_Waitany consumed 10.91% . Upon examining the coefficient of variation for all 12 runs of the same benchmark, we notice that there is a substantial amount of fluctuation. The variation increases dramatically at 8 cores and is larger, in general, for EC2 than in FG.

As is seen from Figures 1,2,3 the variation also depends on the solver/preconditioner used. This is an indication that no single solver/preconditioner combination will work best for all problem sizes and all core counts. There does appear to be a "sweet-spot" at 4 cores for most problem sizes with most solver/preconditioner combinations that does not suffer from large variations in runtime on India (FG); however the same sweet-spot suffers from large fluctuation in the runtime (up to 150% of the mean) on EC2. This presents interesting challenges when scheduling jobs on mixed cloud environments (e.g., FG and EC2).

It is also worth mentioning that the variation in performance on a single core is more pronounced on India than on EC2. It can reach 25% of the mean on a 20x20x20 problem with a GMRES solver and no preconditioner and upwards of 60% of the mean with a 40x20x20 problem with a GMRES solver and BJacobi preconditioner. The root cause of this problem is not immediately apparent, and warrants further investigation. We have begun running SKaMPI [9] benchmarks on both EC2 and India; early indications are that there is a large variation in performance for collective MPI operations and a significantly smaller variation for point-to-point communication.

VI. Conclusion and Future Work

We have investigated the performance of a real world application on EC2 and FutureGrid and performed detailed performance fluctuation studies. These fluctuations need to be analyzed with more detailed, function–cost benchmarks running at the same time with host system profilers. Reducing the fluctuation on a single instance is mandatory step to expanding our work to large core counts and running MPI applications efficiently across several instances.

Acknowledgements

The research presented in this paper is supported in part by National Science Foundation via grants numbers IIP 0758566, CCF-0833039, DMS-0835436, CNS 0426354, IIS 0430826, and CNS 0723594, by Department of Energy via grant numbers DE-FG02-06ER54857 DE-FG02- 04ER46136 (UCoMS), and was conducted as part of the NSF Center for Autonomic Computing at Rutgers University. Experiments on the Amazon Elastic Compute Cloud (EC2) were supported by a grant from Amazon Web Services and CCT CyberInfrastructure Group grants. We thank FutureGrid for early user access to their systems; we also thank them for system support and help (in particular Joe Rimkosvky). SJ and MP would like to acknowledge the e-Science Institute, Edinburgh for supporting the Research theme on Distributed Programming Abstractions.

References

[1] J. Ekanayake, X. Qiu, T. Gunarathne, S. Beason, and G. Fox, "1 high performance parallel computing with cloud and cloud technologies."

[2] L. Youseff, R. Wolski, B. Gorda, and C. Krintz, "Evaluating the performance impact of xen on mpi and process execution for hpc systems," in *in 2nd International Workshop on Virtualization Technology in Distributed Computing (VTDC)*, p. 1.

[3] Futuregrid. http://www.futuregrid.org/.

[4] H. Kim, Y. el Khamra, S. Jha, and M. Parashar, "An autonomic approach to integrated hpc grid and cloud usage," in *e-Science, 2009. e-Science '09. Fifth IEEE International Conference on*, Dec. 2009, pp. 366–373.

[5] ——, "Exploring application and infrastructure adaptation on hybrid grid-cloud infrastructure," in *1st Workshop on Scientific Cloud Computing in conjunction with 19th ACM International Symposium on High Performance Distributed Computing*, June 2010, pp. 402–412.

[6] G. Wang and T. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *INFOCOM, 2010 Proceedings IEEE*, mar. 2010, pp. 1 –9.

[7] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," 2001, http://www.mcs.anl.gov/petsc.

[8] D. Skinner, "Integrated performance monitoring," http://ipm-hpc.sourceforge.net/overview.html.

[9] "Skampi 5 benchmark for mpi implementations," http://liinwww.ira.uka.de/ skampi/index.html.