Evaluating High Performance Computing on the Windows Azure Platform

Eduardo Roloff, Francis Birck, Matthias Diener, Alexandre Carissimi, Philippe O. A. Navaux
Informatics Institute
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
{eroloff, fbmoreira, mdiener, asc, navaux}@inf.ufrgs.br

***Abstract*—Using the Cloud Computing paradigm for High-Performance Computing (HPC) is currently a hot topic in the research community and the industry. The attractiveness of Cloud Computing for HPC is the capability to run large applications on powerful, scalable hardware without needing to actually own or maintain this hardware. Most current research focuses on running HPC applications on the Amazon Cloud Computing platform, which is relatively easy because it supports environments that are similar to existing HPC solutions, such as clusters and supercomputers.**

In this paper, we evaluate the possibility of using Microsoft Windows Azure as a platform for HPC applications. Since most HPC applications are based on the Unix programming model, their source code has to be ported to the Windows programming model in addition to porting it to the Azure platform. We outline the challenges we encountered during porting applications and their resolutions. Furthermore, we introduce a metric to measure the efficiency of Cloud Computing platforms in terms of performance and price. We compared the performance and efficiency of running these benchmarks on a real machine, an Amazon EC2 instance and a Windows Azure instance. Results show that the performance of Azure is close to the performance of running on real machines, and that it is a viable alternative for running HPC applications when compared to other Cloud Computing solutions.

***Keywords*-High-performance computing (HPC); Performance Evaluation; Benchmark; Efficiency; NAS;**

I. Introduction

In the cloud computing field, the development of cloud services by different companies has impacted the scientific community. Several companies, such as Amazon, Google, IBM and Microsoft, are providing scalable cloud computing services that can potentially replace private cluster and grid systems [1]. Among these, the Amazon cloud has been extensively evaluated in the scientific community [2][3]. The reason for this popularity is that it uses the *Infrastructure as a Service (IaaS)* model [4]. In this model, developers have more flexibility to adapt the programming and execution environment to the needs of their applications. Deployment on IaaS is made using virtual machines, where the developer can choose the operating system and support libraries, among others.

On the other hand, the Windows Azure platform is based on the *Platform as a Service (PaaS)* model, and evaluating its performance using standard HPC applications and benchmarks presents some challenges. In the PaaS model, the developer does not have control over the cloud infrastructure and the operating system, and has to use the tools and libraries supplied by the service provider.

More specifically, the PaaS model leads to two challenges for the execution of HPC applications on Azure. First, since the source code of HPC applications is usually aimed at operating systems compatible with Unix, it needs to be converted to the Windows programming model supported by Azure. This is difficult in many cases, especially if support libraries (such as parallelization APIs or I/O libraries) are needed or if the build system or compiler are not compatible with Windows. Second, there are additional modifications necessary to adapt the application to the requirements of the Azure PaaS, such as the transformation of the source code into a library executable on Azure. For these reasons, there are few studies evaluating the performance of HPC applications on Windows Azure. Previous research in HPC on Azure focuses on writing new applications specifically for Azure [5] [6], which leads to duplication of work and lack of comparability between different solutions.

The goal of this paper is to port a set of well-known HPC benchmarks, the *NAS Parallel Benchmarks (NPB)* [7], to the Azure PaaS and evaluate their performance. Our work has three main contributions. First, we analyze the complexity of porting existing HPC applications to the Azure platform, detailing the challenges and their resolutions. Second, we introduce a metric which compares the price and performance of different cloud computing solutions, which leads to the notion of *efficiency* of a cloud computing solution. Finally, we evaluate the performance and efficiency of Azure by comparing it to a real machine and to an Amazon IaaS cloud.

This paper is organized as follows. In Section II we give an overview about related work in this area and compare them to our work. Section III describes and explains the porting process of HPC applications to the Azure platform. Moreover, we detail the challenges we encountered during the process and their solutions. We introduce an efficiency metric for cloud systems and present the evaluation methodology of the performance experiments in Section IV. The results of our experiments are shown and discussed in Section V. In Section VI we conclude our work with discussions on the future of cloud computing on Azure based on our findings as well as mentioning some ideas for future research.

II. Related Work

Related work in this area can be divided into three categories: research on running HPC applications on Windows Azure, performance evaluation of clouds, and migration of applications into the cloud. In this section, we will give a brief overview over this research and compare it to our work.

Lu et al. [6] and Li et al. [5] evaluate the possibility of writing scientific applications to run on the Azure platform. They conclude that it is a viable solution for scientific applications that involve large data sets. In contrast to these works, we do not write new applications in Azure, but rather port existing HPC applications to the Azure platform and evaluate the possibility and the difficulties of doing so.

The article of Walker [8] evaluates the performance degradation of Amazon EC2 instances. He compares the performance of the NAS Parallel Benchmarks with their performance on a real machine and analyzes the network efficiency by executing the *mpptest* [9] MPI benchmark. Both computing and network performance proved to be challenges for the cloud solution. In this paper, we perform similar experiments with the Azure cloud and compare the performance and efficiency to the Amazon cloud and a real machine.

Deelman et al. [2] evaluate the cost of Amazon EC2 by porting a real-life astronomy application to the cloud and execute it using different resource provisioning plans. They conclude that the cloud is a cost-effective option since the scientific application provider does not need to buy an entire cluster for a few runs of the application. Many clusters are underused as the hardware quickly becomes obsolete. The cloud solves this problem as it is a responsibility of the cloud provider to keep upgrading the hardware and provide an up-to-date service to the users. In our research, we compare Azure and EC2 and evaluate the relation between cost and performance of both clouds.

The performance and performance variability of the EC2 cloud is evaluated by Jackson et al. [10], comparing the cloud system to real clusters by running several scientific applications. Their results show that performance degradation and variability of the EC2 are very high, citing the speed of the interconnections as a major problem.

Ekanayake et al. [11] evaluate the performance of a new MapReduce implementation, CLG-MapReduce, and compare it to the existing implementations Hadoop and Dryad on cloud platforms. Furthermore, they compare the performance to a more traditional MPI implementation. Their results show that for various applications, CLG-MapReduce has a similar performance to MPI. However, almost all evaluated applications, such as the EP benchmark from NAS, have little or no communication between the threads. As communication is an important part of HPC applications, we evaluate a wider range of applications, with different amounts of communication and varying communication patterns.

Li et al. [12] compare four commercial cloud providers: Amazon EC2, Windows Azure, Google AppEngine and Rackspace CloudServers. They measure the performance of several components of the cloud solutions, including computing, network, database and storage. They conclude that no cloud solution is best overall in terms of performance, but that each cloud excels in different areas. The focus of our work is the single instance performance, as it is the primary building block for larger HPC applications.

As an example of migration of applications to the cloud, Ward et al. [13] study the automatic migration of commercial applications to the IBM cloud. They create a framework which automates many of the steps needed, and analyze the time and work savings compared to traditional server deployment. The IBM cloud and the studied applications are based on Linux, which simplifies the migration process.

In contrast to the work presented here, our work focuses on evaluating Windows Azure as a platform for HPC and compare it to other solutions. We port largely unmodified HPC applications to the Azure platform, and introduce a metric to compare performance and cost of different cloud computing solutions.

III. Porting HPC Applications to the Azure PaaS – Challenges and Solutions

In this section, we explain the porting process for an HPC application to the Azure PaaS platform. First, we give a short overview of the platform. Afterwards, we explain in detail the challenges and solutions of porting the application to the Win32 programming model and to the Azure platform.

A. Overview of the Azure Platform

The Windows Azure Platform provides a PaaS [4] that allows the execution of applications using the Azure operating system. This operating system is composed of three components: Compute, Storage and Fabric. As we are using only the Compute component, we explain this component in more detail here.

The Compute component provides an interface to allow programming, hosting and executing programs using instances of the Web Role, Worker Role and VM Role. The Web Role is the instance that provides the web interface to the user, allowing interactivity with the OS in order to program and execute code. The Worker Role is the instance that actually runs the code. By allocating multiple Worker Role instances it is possible to scale the execution. The VM Role allows users to create IaaS systems. However, the IaaS service is still in the beta development stage, and is unavailable to regular users.

As a consequence of using the Azure PaaS, several cloud features, such as the programming model and the operating system, are predetermined and can not be influenced by the developer. In the case of Azure, the programming model is the *Win32 model* and the operating system is the aforementioned Azure OS. As most existing HPC applications are programmed using the Unix programming model, porting such an application to Azure consists of two steps: modifying the source code to be compatible with Win32 and making the program able to execute on Azure. Therefore, a port of an application to Azure is not completely transparent to the developer, as modifications to the source code of the application are required.

In this study we try to do the smallest possible amount of changes to the source code. We organize our research into two parts. First, we port the applications to run on Win32 and then to the Azure platform, while detailing the challenges encountered and their resolutions, as porting can become a complex process [13]. We use the NAS Parallel Benchmarks [7] as an example for the porting process in the rest of this section.

B. Porting to the Win32 Programming Model

Porting the applications to the Win32 programming model presented three challenges: the build system, the Fortran compiler, and differences between the system calls of Unix and Win32, which manifested themselves in the NAS benchmarks in the timing related functions.

The most common environment to write Windows applications is Microsoft Visual Studio, which uses a different build system from traditional Unix makefiles used in the NAS benchmarks. As adapting the existing makefile-based build system was considered too complex, we recreated a similar build system using the project functions of Visual Studio, adapting the build parameters without changing the source code itself.

The second issue is that Visual Studio does not include a native Fortran compiler. Fortran is widely used in the scientific community and is the language of choice for many HPC applications. Eight out of the ten NAS benchmarks are written in Fortran. After some research we chose the Intel Visual Fortran Composer XE for Windows, which integrates well with Visual Studio and compiles the source code of the NAS benchmarks written in Fortran correctly.

The most work intensive issue when porting to Win32 arises when there are incompatibilities between system library functions. In the NAS benchmarks, the included timing functions did not work as they use system calls that are only available on Unix. Therefore, the entire timing mechanism had to be rewritten to be compatible with Win32.

After these steps we were able to successfully run all the NAS benchmarks on Windows, and we verified that there isn't a significant performance difference when compared to a Linux operational system. Achieving this, we then proceeded to port the benchmarks to the Azure Platform.

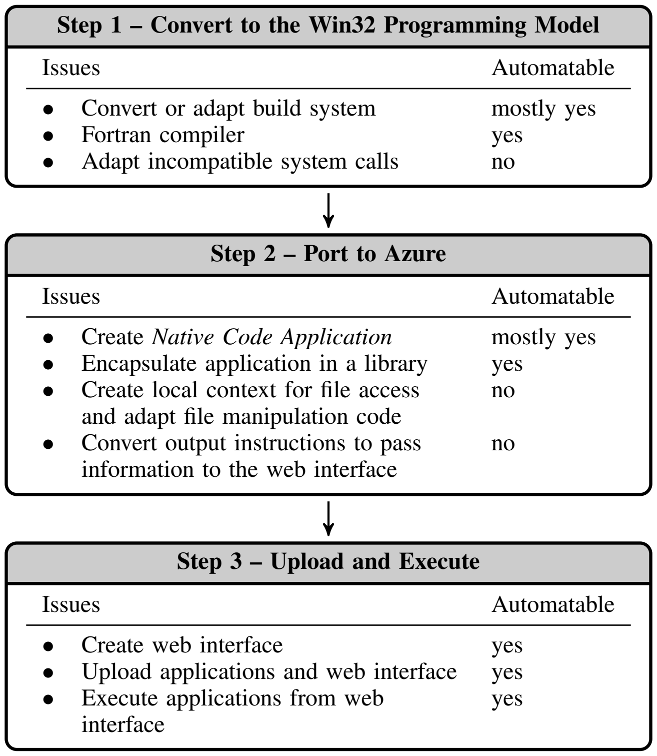
C. Porting to the Azure Platform

An application in the Azure PaaS needs to be web based, with an interface (the Web Role) and a computing node (the Worker Role). To run the benchmarks on Azure, the source code of the applications has to be ported to this model. As Azure does not support Fortran and C natively, creating a *Native Code Application* is required. In this way, any application written in a language supported by Windows can be executed on Azure. To create the native code application, it is necessary to create a project with the web interface and start the execution of the benchmark from this interface.

As the application's code is hosted in the native code application, the benchmarks need to be encapsulated in a library (DLL). This is called the native code execution. Microsoft provides online documentation [1] to help with the creation of such a library. In order to use the benchmarks as DLLs, it is necessary to change the original main functions to an exported function.

File system access also presents a challenge on Azure. The applications run within a virtual server which does not have native access to a file system. Therefore, it is required to create a virtual disk space with a specified size to hold the files. However, this local disk space cannot be accessed directly as its path changes for every execution. To solve this

[1] http://msdn.microsoft.com/en-us/hh351535



Step 1 – Convert to the Win32 Programming Model

Issues	Automatable
• Convert or adapt build system	mostly yes
• Fortran compiler	yes
• Adapt incompatible system calls	no

Step 2 – Port to Azure

Issues	Automatable
• Create *Native Code Application*	mostly yes
• Encapsulate application in a library	yes
• Create local context for file access and adapt file manipulation code	no
• Convert output instructions to pass information to the web interface	no

Step 3 – Upload and Execute

Issues	Automatable
• Create web interface	yes
• Upload applications and web interface	yes
• Execute applications from web interface	yes

Figure 1. Summary of the steps necessary to execute an HPC application on Windows Azure.

problem, the application needs to access the *local context* to get information about the path to the virtual disk. This path is then included in the full path used in the `fopen` system call and the application is able to work with files using the regular functions for file manipulation. From the NAS benchmarks, the DC benchmark reads and writes files during the execution. Thus, for DC, the entire file manipulation code had to be changed to include the full file path of the virtual hard drive.

In Azure, no output console for the *stdout* output from applications exists, so the console print functions do not work. To obtain the output, it is necessary to create local variables in the web interface and pass them as a reference to the DLLs. We used these local variables to get information about execution time, available threads and a flag for successful execution.

With all the modifications done, it is necessary to upload the application and the web interface to the Azure platform, where it can be executed through the web interface. After the modifications, all of the NAS benchmarks, except FT, were able to run on the Azure platform. FT contains many procedure calls in the source code and was terminated by the Azure platform due to insufficient stack size. We found no way to change the maximum stack settings of the Azure instance.

Figure 1 shows a summary of the steps necessary to transform an HPC application to be able to execute it on the Azure platform, as well as showing which of the steps can possibly be automated. Though several of the steps could be automated, many of the more difficult steps (mostly those involving modifications to the source code of the application) have to be done manually by the developer for each application. In the next sections, we will evaluate the performance of the NAS applications running on Azure and compare it to other systems.

IV. Evaluation Environments and Methodology

To compare the performance and price of cloud solutions, we introduce a new metric in this section. This metric presents a notion of efficiency of cloud solutions in terms of price and performance. Afterwards, we present the environments in which we performed experiments and detail their configurations. We evaluated the performance of the Azure platform and compared it to an Amazon EC2 IaaS cloud and a real machine. Finally, we present the detailed characteristics of the NAS benchmarks used in the experiments.

A. Comparing Cloud Efficiency with a new Metric

Comparisons of cloud solutions need to take into account two metrics, the performance of execution as well as the cost of execution, as these matter most to the user. As the performance and price of clouds differ widely, it is not sufficient to use only one of these metrics to provide a fair comparison. Combining the cost and performance metrics leads to the notion of *efficiency* of a cloud solution. We therefore introduce a metric to express this efficiency.

We calculate this metric as follows. After measuring the execution time of an application, we calculate how many times this application can be executed in one hour. This time frame is motivated by the fact that the billing of most clouds is done on an hourly basis as well. Afterwards, we divide the value obtained by the hourly cost of the cloud system. Thus we get a representation of the number of executions per dollar for a specific application. This metric will be used to compare the cloud solutions in Section V.

B. Machines

The environments we used in our experiments, two cloud systems and one real machine, are described in this section.

1) Windows Azure PaaS Cloud: Azure offers five different Compute instance sizes. To get the most relevant data for HPC, we selected the instance with the highest performance for our experiments. This *Extra-large Compute* instance consists of eight cores running at 1.6 GHz and 14 GByte of memory. Running this instance costs USD 0.96 per hour[2].

2) Amazon Elastic Compute IaaS Cloud: The Amazon *Elastic Compute Cloud (EC2)* is an IaaS cloud and currently supports a variety of operating systems, including several Linux distributions, Solaris and Windows Server. The EC2 cloud uses Xen [14] as the virtualization system. The hourly charge for a running virtual machine are based on the resources allocated to the machine (CPU cores, memory and storage) as well as license fees for the pre-installed software.

We selected the *Hi-CPU Extra Large On-Demand* instance, which is the instance whose specifications match the specifications of the Azure instance the closest. To make the comparison between different instances easier, Amazon uses *Compute units* to describe the performance of an instance. One EC2 compute unit is the equivalent of a 2007 Intel Xeon core running at 1.0 – 1.2 GHz, or a 2006 Intel Xeon core running at 1.6 GHz. The selected instance consists of eight cores with 2.5 EC2 compute units each. This indicates that the cores are running at an equivalent speed of about 2.5 GHz – 3.0 GHz. Running this instance with the Windows 2008 Server operating system cost USD 1.16 per hour[3].

3) Real machine: To compare the cloud computing platforms and evaluate their overhead, we ran the benchmarks on a real machine whose specifications match the hardware used in the cloud solutions as closely as possible. This real machine consists of two Intel Xeon E5530 processors, each containing four cores running at 2.33 GHz with one thread per core and 12 GByte of main memory. To obtain a fairer comparison, we downgraded the frequency of the processors

[2]http://www.windowsazure.com/en-us/pricing/calculator/advanced/
[3]http://aws.amazon.com/ec2/#pricing

Table I
COMPARISON OF THE SYSTEMS USED IN THE EVALUATION.

	Windows Azure	Amazon EC2	Real machine
Service model	PaaS	IaaS	n/a
Processor generation	not specified	2006 or 2007 Intel Xeon	Intel Xeon E5530 (Nehalem, 2008)
Processor Speed (GHz)	1.6	$\sim 2.5 - 3.0$	1.6
Number of cores	8	8	8
Memory (GByte)	14	7	12
Price per hour (USD)	0.96	1.16	n/a

to 1.6 GHz, which is the frequency of the processor in the Azure platform.

Table I summarizes the relevant properties of the execution environments for comparison.

C. Benchmarks used in the Evaluation

To evaluate the performance, we used the OpenMP versions of the *Numerical Aerodynamic Simulation Parallel Benchmarks (NPB)* [7], in the 3.3.1 version. This benchmark suite is composed of several applications performing numerical methods used in aerodynamic simulations for scientific computing. The following ten applications are part of the suite: BT (Block Tridiagonal), CG (Conjugate Gradient), DC (Data Cube Operator), MG (Multigrid), EP (Embarrassingly Parallel), SP (Scalar Pentadiagonal), LU (Lower and Upper triangular system), IS (Integer Sort), FT (Fourier Transform), UA (Unstructured Adaptive).

BT, SP and LU have their parallelism expressed as the division of the spatial domain, while sharing data in the borders of each subdomain, showing linear access to data. The CG and UA present random data access characteristics. The MG application works on contiguous data, varying the access pattern in each computing step, going from unaligned accesses to aligned accesses as the grid is refined. Thus, this application can be considered as an intermediate between the regular access applications (BT, LU and SP) and the irregular access applications (CG and UA).

EP is completely parallel and has few memory accesses. The performance of this application can be used as a reference for the peak computational performance of a system. IS only uses integer calculations and not floating point calculations as the other benchmarks. It also has a medium amount of communication between the threads.

Due to the problems with the stack size mentioned in Section III-C, we did not include FT in our experiments. DC focuses on the performance of file system access, creating and modifying large data files (up to 20 GByte). As we are focused on evaluating the computing part only, we did not include DC as well.

Table II
OVERVIEW OF THE NAS BENCHMARKS USED IN THE EVALUATION.

Name	Description	Focus	Language	Memory Usage
BT	Block Tridiagonal	Floating point performance	Fortran	44 MByte
CG	Conjugate Gradient	Irregular communication	Fortran	46 MByte
EP	Embarrassingly Parallel	Floating point performance	Fortran	9 MByte
IS	Integer Sort	Integer performance	C	66 MByte
LU	Lower and Upper Triangular	Regular communication	Fortran	41 MByte
MG	Multigrid	Regular communication	Fortran	431 MByte
SP	Scalar Pentadiagonal	Floating point performance	Fortran	50 MByte
UA	Unstructured Adaptive	Irregular communication	Fortran	32 MByte

From the eight applications we chose to run, seven are written in Fortran and one (IS) in C. All benchmarks except IS execute floating point operations of double precision (64 bits). These benchmarks focus on four different usage patterns: Floating point performance, Integer performance, Regular and Irregular Communication.

We chose the *A* input size for the NAS benchmarks, which represents a medium input size. Table II contains an overview of the characteristics of the NAS benchmarks. It is important to note that the memory usage of each benchmark is lower than the memory size of each system used in our evaluation (cf. Table I). Therefore, the different memory sizes do not influence the performance of the benchmarks.

V. RESULTS

This section presents the results of our experiments with the NAS benchmarks on the three execution environments. First, we show the results of the performance experiments and analyze them. In the second part, we use the metric introduced in Section IV-A to compare the efficiency of the cloud solutions.

A. Performance Results

The performance results of the benchmarks presented in Section IV-C are shown in Figures 2 and 3. The Figures show the average execution time of 50 runs and the confidence interval for a confidence level of 95% in a Student's t-distribution. Table III contains the performance degradation of the cloud solutions compared to the real machine, in percent, as well as the average degradation over all benchmarks.

In most cases, the benchmarks show lower performance on the cloud solutions compared to the real machine. A small

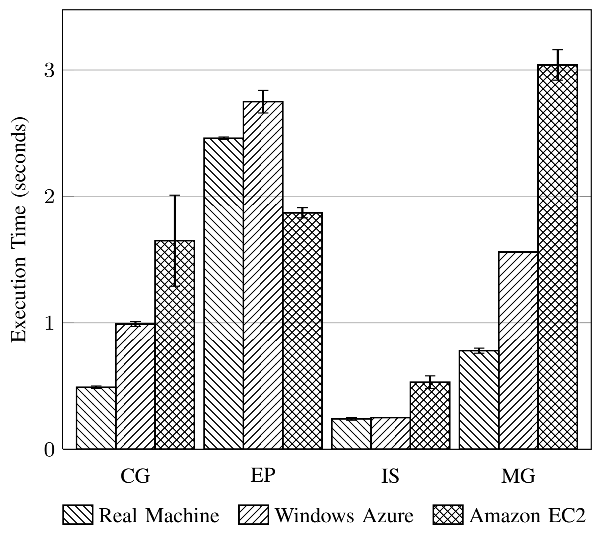


Figure 2. Execution time of CG, EP, IS and MG

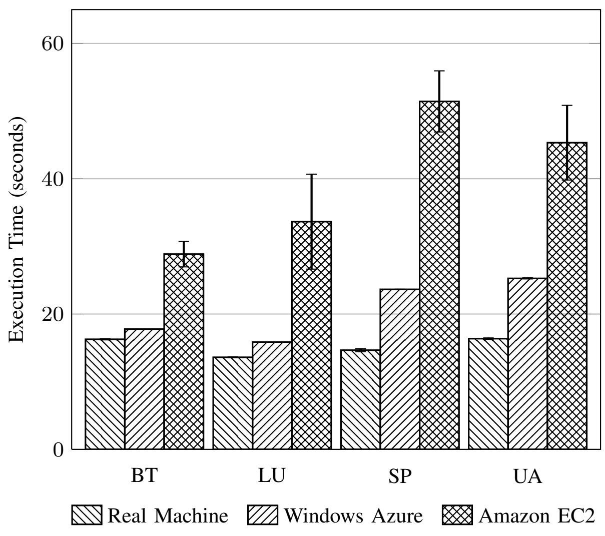


Figure 3. Execution time of BT, LU, SP and UA

performance degradation is expected due to the virtualization solution. Regarding the amount of degradation on Azure, it is possible to separate the benchmarks into two groups. The first group, consisting of CG, MG, SP and UA, show a large degradation of 35% to 50%. The second group, consisting of EP, IS, BT and LU, show a much smaller degradation of 3% to 15%. The reasons for these results are as follows.

The IS, BT and LU benchmarks focus on integer and floating point performance and also have a medium amount of communication. Their performance on Azure is close to the performance on the real machine. On EC2 however, the performance degradation is much larger, up to 60%.

EP shows different behavior from all other benchmarks. The performance of the EC2 cloud surpasses both the Azure cloud and the real machine. As it uses an embarrassingly parallel algorithm with very little memory usage and communication, its performance depends to a large extent on the FPU performance of the processors and not on the memory performance. From the EP result we can therefore conclude that the EC2 cloud has a faster processor model than the Azure cloud, as could be expected from the calculation of the frequency of the processors in the cloud in Section IV-B2. The CG, MG, SP and UA benchmarks present a large performance degradation. Three of these benchmarks, CG, MG and UA, focus on communication. The SP benchmark, although focusing on FPU performance and being similar in structure and operation to BT, has a much larger amount of communication between the threads [15]. Despite the high performance degradation on Azure, the degradation on EC2 is even higher, reaching up to 74%. These results lead to three conclusions for the interconnections: their performance is an issue in cloud systems, the interconnection on EC2 is slower than on Azure, and they are very relevant for the performance of HPC applications.

Another important result is the variability of the execution time of the benchmarks. Low variability is an important factor for HPC, as it leads to a higher predictability of experiments, both for the time it takes and the cost. From the confidence interval shown in the graphs, it is easy to see that performance is very predictable on the real machine as there is practically no variability for all benchmarks. On Azure, the results are very similar. All benchmarks except EP have a negligible variability on these architectures.

On the EC2 cloud however, the variability is much higher than on Azure in general. CG, LU, SP and UA have a large variation, while IS, MG and BT show small to medium amount of variation. EP is again showing a different behavior from the other benchmarks, as its variability is lower than Azure's. As EP has very little communication, the scheduling of the threads does not matter for the performance and therefore does not cause a great variation in the execution time.

High variability is generally a sign of problems in the scheduler [16]. In a cloud system, it can be caused by scheduling strategies of the virtualization solution. For example, scheduling the threads of an application on different NUMA nodes can lead to performance degradation due to higher memory access and communication latencies.

Table III
PERFORMANCE DEGRADATION OF THE CLOUD SOLUTIONS COMPARED TO THE REAL MACHINE, IN PERCENT.

	BT	CG	EP	IS	LU	MG	SP	UA	Avg.
Azure	8.5	50.7	10.5	3.2	14.2	50.1	38.0	35.3	26.3
EC2	43.6	70.5	-32.1	55.1	59.6	74.4	71.5	63.9	50.8

Table IV
NUMBER OF EXECUTIONS PER HOUR AND EFFICIENCY METRIC OF WINDOWS AZURE AND AMAZON EC2.

	Executions / hour		Efficiency Metric		
App.	Azure	EC2	Azure	EC2	Difference
BT	202	125	211	108	49.0 %
CG	3,636	2,182	3,788	1881	50.3 %
EP	1,309	1,925	1,363	1660	-21.7 %
IS	14,400	6,792	15,000	5856	61.0 %
LU	227	107	236	92	61.0 %
MG	2,307	1,184	2,404	1021	57.5 %
SP	152	70	159	60	61.9 %
UA	142	79	148	68	53.9 %

To summarize, the performance results show that raw processor performance on EC2 is superior to Azure. However, EC2 suffers from performance degradation due to the interconnections and scheduler problems. For HPC, this means that there is no perfect cloud solution, but that the solution has to be carefully chosen according to the usage pattern of each application.

B. Comparing Efficiency

We also calculated the efficiency of the cloud solutions in terms of price and performance as introduced in Section IV-A. Table IV shows the number of executions per hour, the efficiency metric of each benchmark and the difference between the metric for the two solutions.

We used the instance prices of USD 0.96 and USD 1.16 per hour, for Azure and EC2 respectively. We do not include the time it takes to boot an instance, as this time is negligible over a longer period of execution. Our results indicate that for most of the NAS benchmarks, Azure offers more efficient single instances than EC2. The only exception is the EP benchmark. Despite the higher instance cost of EC2, it is still more efficient on EC2 than Azure. This indicates that EC2 is more efficient for this kind of application, which echoes the result from the last section where pure FPU performance is important and there is little communication between the threads.

VI. CONCLUSIONS AND FUTURE WORK

For the HPC community, the cloud idea has offered an efficient way to eliminate the need of owning a cluster or grid. Cloud systems ensure that the user does not need to concern himself with buying, maintaining and upgrading physical machines, and can focus on actually running applications.

In this paper, we evaluated the Windows Azure platform as a platform to run general HPC applications. We showed how to port applications written for the UNIX programming model to Azure and compared their performance to a real machine and an Amazon EC2 instance with similar characteristics. We measured the execution time and examined the efficiency of the cloud solutions in terms of performance and cost with a new metric.

Our results show that Azure offers good performance for most benchmarks tested. Compared to the real machine, it has a low overhead when the application communicates little, and a moderate overhead with large amounts of communication. This indicates that its performance is limited by the speed of the interconnections. Compared to the EC2 cloud, Azure has generally shown better performance due to better interconnections. Applications that focus mostly on raw processing however proved to be faster on EC2. Efficiency was also shown to be higher on Azure. Despite the challenges to port applications, Azure has therefore proven to be a realistic platform for HPC.

For the future, we intend to port benchmarks and applications using the MPI programming model to Azure and evaluate their performance. We also aim to create a program that can help with the porting process to Azure.

ACKNOWLEDGMENT

This work was partially supported by CNPq, FAPERGS and RNP. Microsoft provided a free Azure instance for the experiments.

REFERENCES

[1] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, may 2009, pp. 1 –12.

[2] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: The montage example," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, nov. 2008, pp. 1 –12.

[3] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 931 –945, june 2011.

[4] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Tech. Rep., 2011. [Online]. Available: http://www.mendeley.com/research/the-nist-definition-about-cloud-computing/

[5] J. Li, M. Humphrey, D. Agarwal, K. Jackson, C. van Ingen, and Y. Ryu, "escience in the cloud: A modis satellite data reprojection and reduction pipeline in the windows azure platform," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, april 2010, pp. 1 –10.

[6] W. Lu, J. Jackson, and R. Barga, "Azureblast: a case study of developing science applications on the cloud," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 413–420. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851537

[7] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS parallel benchmarks and its performance," *NASA Ames Research Center, Technical Report NAS-99-011*, 1999.

[8] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," *Usenix Login*, vol. 33, no. 5, pp. 18–23, 2008.

[9] W. Gropp, "Reproducible measurements of MPI performance characteristics," *Recent Advances in Parallel Virtual Machine and*, 1999. [Online]. Available: http://www.springerlink.com/index/PE2J40W56L9GM1Y9.pdf

[10] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, and N. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 30 2010-dec. 3 2010, pp. 159 –168.

[11] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies," *Cloud Computing*, pp. 20–38, 2010.

[12] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th annual conference on Internet measurement*. ACM, 2010, pp. 1–14.

[13] C. Ward, N. Aravamudan, K. Bhattacharya, K. Cheng, R. Filepp, R. Kearney, B. Peterson, L. Shwartz, and C. Young, "Workload migration into clouds challenges, experiences, opportunities," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, july 2010, pp. 164 –171.

[14] P. Barham, B. Dragovic, K. Fraser, and S. Hand, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=945462

[15] C. Coarfa, Y. Dotsenko, J. Eckhardt, and J. Mellor-Crummey, "Co-array fortran performance and potential: An npb experimental study," *Languages and Compilers for Parallel Computing*, pp. 177–193, 2004.

[16] A. Fedorova, M. Seltzer, and M. Smith, "Cache-fair thread scheduling for multicore processors," *Division of Engineering and Applied Sciences, Harvard University, Tech. Rep. TR-17-06*, 2006.