Scientific Programming 19 (2011) 121–132
DOI 10.3233/SPR-2011-0323
IOS Press

Early observations on the performance of Windows Azure

Zach Hill *, Jie Li, Ming Mao, Arkaitz Ruiz-Alvarez and Marty Humphrey
Department of Computer Science, School of Engineering, University of Virginia, Charlottesville, VA, USA

Abstract. A significant open issue in cloud computing is the real performance of the infrastructure. Few, if any, cloud providers or technologies offer quantitative performance guarantees. Regardless of the potential advantages of the cloud in comparison to enterprise-deployed applications, cloud infrastructures may ultimately fail if deployed applications cannot predictably meet behavioral requirements. In this paper, we present the results of comprehensive performance experiments we conducted on Windows Azure from October 2009 to February 2010. In general, we have observed good performance of the Windows Azure mechanisms, although the average 10 min VM startup time must be accounted for in application design. We also present performance and reliability observations and analysis from our deployment of a large-scale scientific application hosted on Azure, called ModisAzure, that show unusual and sporadic VM execution slowdown of over 4× in some cases and affected up to 16% of task executions at times. In addition to a detailed performance evaluation of Windows Azure, we provide recommendations for potential users of Windows Azure based on these early observations. Although the discussion and analysis is tailored to scientific applications, the results are broadly applicable to the range of existing and future applications running in Windows Azure.

Keywords: Cloud computing, eScience, performance, Windows Azure

1. Introduction

Cloud computing has burst onto the high performance computing scene in recent years and has established itself as a viable alternative to customized HPC clusters for many users who do not have the resources – either time or money – to build, configure, and maintain a cluster of their own. The ability to pay only for resources that are utilized and the apparent ease by which resources can be expanded, deployed, and removed from service are very attractive. Many eScience developers are increasingly looking to create data-intensive applications that have highly variable resource requirements over time and can take advantage of the pay-as-you-go cost model to lower costs, increase scale, and decrease deployment time.

As more cloud providers and technologies enter the market, developers are faced with an increasingly difficult problem of comparing various offerings and deciding which vendor to choose for deploying an application. One critical step in the process of evaluating various cloud offerings is determining the performance of the services offered and how those match the requirements of the application. Even in situations where other factors ultimately dominate the choice regarding potential cloud platforms (e.g., cost per unit time of a virtual machine or application-hosting environment in the particular cloud), it is important to consider performance ramifications of design decisions to ensure maximum value of cloud applications.

The purpose of this paper is to provide a quantitative analysis of the performance of the Windows Azure Platform [18] using micro-benchmarks as well as to provide insights gained developing and deploying a large-scale scientific application for processing satellite imagery in Windows Azure: ModisAzure. We present the results of performance experiments we conducted on Windows Azure from October 2009 to February 2010. Through our existing collaborative partnership with Microsoft Research, we were able to evaluate the services at scales not readily available to users of the early Community Technology Preview (CTP) release – up to 192 concurrent instances. Our methodology is to assume that Windows Azure has already been chosen as the target cloud for whatever reason, and that now the developer is facing the challenge of architecting his/her cloud application to accommo-

*Corresponding author: Zach Hill, Department of Computer Science, School of Engineering, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904-4740, USA. Tel.: +1 434 982-2200; Fax: +1 434 982-2214; E-mail: zjh5f@cs.virginia.edu

date performance considerations. In other words, to maintain a reasonable scope of this research effort (and this paper), we do not directly compare performance of Windows Azure to other clouds. We also note that our collaboration with Microsoft Research does not include us having access to implementation or operational details of the Windows Azure Platform, so we treat the service as a black-box.

The Windows Azure Platform is composed of several services: Windows Azure, SQL Azure Database, SQL Azure Reporting, SQL Azure Data Sync, AppFabric, Content Delivery Network (CDN) and Windows Azure Connect (also known as the Azure Virtual Network). This paper focuses on Windows Azure, which encompasses both compute resources and scalable storage services. Our evaluation of the Windows Azure service begins with the performance of its compute resources and its three primary storage services: blobs, queues and tables. Because these are the basic storage components that scalable Windows Azure applications are built upon, it is important to understand their performance characteristics as scale increases. We then evaluate virtual machine instantiation time because instance acquisition and release times are critical metrics when evaluating the performance of dynamic scalability for applications. We also present an evaluation of direct instance-to-instance TCP performance as this mechanism provides an alternative to the other storage services for communication between instances that has lower latency. In general, we have observed good performance of the Windows Azure mechanisms, although the average 10 min VM startup time must be accounted for in application design if dynamic resource scaling is required.

We also present some of our experiences developing the ModisAzure eScience application for the Windows Azure platform. ModisAzure is a data-processing system satellite imagery that is deployed at a scale of approximately 200 Azure instances. The application went online in February, 2010 and has been used to execute nearly 2.7 million independent sub-tasks on several terabytes of data in a bag-of-tasks architecture. We specifically describe our experiences with virtual machine performance variation and how we observed random slowdowns of VM execution that led us to terminate execution after $4\times$ the normal execution time. The purpose of this data is to show real-world difficulties encountered in cloud application development and their frequency and impact. Finally, we summarize our experimental data into several specific recommendations for developers using Windows Azure Platform. In these recommendations we address virtual machine instances, the storage services, and our experience in testing and developing cloud applications. Although the discussion and analysis is tailored to scientific applications, the results are broadly applicable to the range of existing and future applications running in Windows Azure.

This work extends previous work [11] with the inclusion of the ModisAzure application sections, and where that work included the evaluation of Azure SQL Services, we have omitted it here due to space constraints. The rest of this paper is organized as follows: Section 2 surveys related work. We start our analysis of the Windows Azure storage services in Section 3. We then discuss the results of our experiments with the Azure computing services in Section 4. Section 5 describes the ModisAzure application very briefly and presents our performance and failure data from its deployment. We discuss the implications of our results for both users and cloud providers in Section 6. Finally, we conclude with Section 7.

2. Related work

In the research community, there is an increasing recognition of both the usefulness and performance concerns of clouds and their underlying technologies. Foster et al. [8] provide an overview of and comparison between grid computing and cloud computing and their architectures, programming models, and management issues. For example, Menon et al. [17] and others [21] evaluated the performance overhead of Xen [3], a software virtualization technology which is a popular choice as the low level virtual machine manager by several cloud providers. Xen has been shown to impose negligible overheads in both micro and macro benchmarks [23]. A higher level analysis is provided by Garfinkel [9], who evaluates some of the cloud services that Amazon provides. Our earlier work compared the performance of cloud platforms with local HPC clusters for scientific applications [12]. Another report examines the feasibility of using EC2 for HPC in comparison to clusters at NCSA [22]. This comparison pits EC2 against high-end clusters utilizing Infiniband interconnects.

There are also studies that focus on a specific scientific application, such as DZero [19] or Montage [5], to evaluate the possibility of migrating existing applications and data to the cloud, based on performance and cost parameters. Workflows [13] and service-oriented

applications [6] have also been the object of study. Another study reports on the possibility of running coupled ocean-atmosphere simulations on EC2 [7]. The research reported in this article complements this earlier work by providing a direct measurement of the mechanisms and APIs of a specific cloud, Windows Azure.

3. Azure storage services

In this section we start our analysis with the performance tests of all three Azure storage services: blob, table and queue. For each service we measure throughput in operations/s or MB/s and the scalability of the service as a function of the number of concurrent clients, among other service-related metrics. For all our tests we use from 1 to 192 concurrent clients. The performance penalty caused by increased concurrency must be taken into account in order to meet the application's requirements, and we provide several data points to help the software developer make decisions about the application's architecture and scale.

For those unfamiliar with Windows Azure and for clarity, Windows Azure offers computation resources (virtual machine instances or VMs) in one of two "roles", or configurations: web or worker. The difference between a "web role" VM instance and a "worker role" VM instance is the software running in the virtual machine and how it is connected to externally. Azure "web role" instances are connected to the outside world through a load-balancer and run Microsoft's Internet Information Services (IIS) to provide web-serving capabilities. The "worker role" instance is not connected to a load-balancer and does not run IIS. For all of our storage service tests the clients we use to test the services are "worker role" instances running within Azure.

3.1. Blob storage

In this section we analyze the performance of both the download and upload of data from the blob service. For our blob download test we use a single 1 GB blob that is stored in Azure. Then, we start a number of worker roles (1–192) that download the same 1 GB blob simultaneously from the blob storage to their local storage. For the upload test, since the different worker roles cannot upload the data to the same blob in Azure storage, a different test is used: the worker role instances will upload the same 1 GB data to the same container in the blob storage, using different blob name. We run the same test three times each day. Although we have collected data for several days at different times, the variation in performance is small and the average bandwidth is quite stable across different times during the day, or across different days.

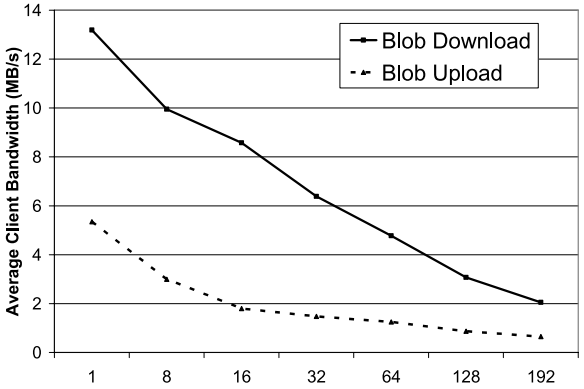

Fig. 1. Average per-client blob download bandwidth as a function of the number of concurrent clients.

The maximum server throughput for the blob download operation was 393.4 MB/s, which was achieved by using 128 clients. For the blob upload operation, the maximum throughput was 124.25 MB/s, which was observed in our experiments with 192 concurrent clients. Figure 1 shows the average client bandwidth as a function of the number of concurrent clients attempting to download the same blob. The bandwidth for 32 concurrent clients is half of the bandwidth that a single client achieves. Using more concurrent clients – up to 128 – increases the total aggregate bandwidth, although this comes with the price of much slower clients. Figure 1 also shows the performance of the upload blob operation, which has a similar curve shape to the download but at about half the bandwidth. For example, average upload speed is only $\sim$0.65 MB/s for 192 VMs and $\sim$1.25 MB/s for 64 VMs. Lower upload bandwidth may be due to internal network policies and/or to constraints on write bandwidth by the complexities involved in creating new blob objects, such as record generation and replication.

3.2. Table storage

A table in Azure is a set of entities with properties, where each property can have various types and the table has no defined schema. We have examined the performance of 4 operations from the Azure Table API: *Insert*, *Query*, *Update* and *Delete*. We have run our experiments with different entity sizes: 1, 4,

16 and 64 kB. In our previous work [11] we showed that for objects larger than 4 kB blob storage gave better performance than table storage in terms operation latency. Thus, we limit our table evaluation to 64 kB-or-less entities. The data types of the fields in the data entity (row) are: {*int*, *int*, *String*, *String}*, not including the row key and partition key that are Strings and required by Azure for uniquely identifying an entity. The last String field is used to set the size of the entity (1 kB, 4 kB, etc.) by filling it with the appropriately sized data. We used up to 192 concurrent clients to study the scalability of each type of table operation. We have found that the shape of the performance curves for different entity sizes are similar, except for some exceptions which are noted below.

For each test case, our experiments performed the following steps: we start with the *Insert* experiment using different numbers of concurrent clients, where each client inserts 500 new entities into the same table partition; after the *Insert* experiment, the table partition includes ∼220 K same-size entities. Next, we perform *Query* operations over the same partition by using a partition key and row key, and each client queries the same entity 500 times. This row and partition key-based query is the fastest query option because they are used for indexing the table. Azure tables also support querying on table properties other than the keys, but we do not evaluate that case here because we are interested in best-case performance. For the *Update* experiment, each concurrent client updates the same entity in the partition and repeats the operation for 100 times. Here we only tested with the *unconditional updates* option as it does not enforce atomicity of each update request, so that different clients can issue update requests to the same table entity at the same time. Finally, in the *Delete* experiment, each client removes the same 500 entities it inserted in the first step of our experiment.

The result of our experiments is summarized in Fig. 2. It presents the data from the point of view of the client, that is, how many operations per second can concurrent clients sustain? For both *Insert* and *Query*, the performance of the clients decreases as we increase the level of concurrency. However, we think that even with 192 concurrent clients we have not hit the maximum server throughput for these two operations. The *Update* and *Delete* tests show more drastic performance declines as we increase the number of clients, though. These two operations has high initial throughput with only 1 client, but then slow down drastically as the number of concurrent clients increases. The maximum throughput for these two services is reached at 8 concurrent clients for the Update operation and 128 for the *Delete* operation.

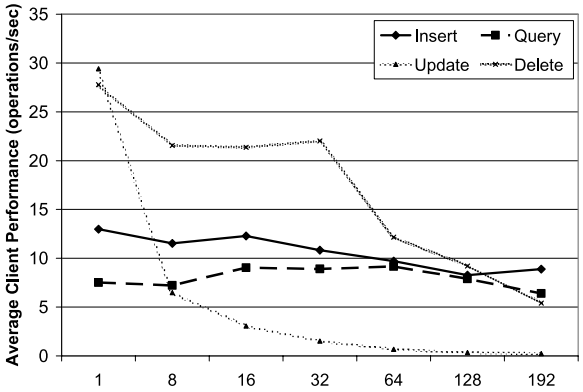

Fig. 2. Average per-client table performance as a function of the number of concurrent clients. Entity size is 4 kB.

Our experiments show that the performance curves for other entity sizes are similar as Fig. 2, except for the following exceptions during the *Insert* and *Delete* tests: For the *Insert* test on 64 kB entities with 192 concurrent clients, only 89 clients successfully finished all 500 insert operations, and the other 103 client have encountered timeout exceptions from the server. With 128 concurrent clients inserting 64 kB entities, only 94 clients successfully finished all 500 operations. This indicates that we may hit the table service capability limit on the above two combinations with large entity size and high concurrency. We have also observed similar behaviors during the *Delete* tests.

3.3. Queue storage

The main purpose of the queue storage service in Windows Azure is to provide a communication facility between web roles and worker roles. For our queue test we use one queue that is shared among several worker roles – from 1 to 192. We examine the scalability of three queue storage operations: *Add*, *Peek* and *Receive*. For each operation we run the test with different message sizes: 512 bytes, 1, 4 and 8 kB. As it was the case with table, the shape of the performance curve for each message size is very similar and we choose to show the results for 512 bytes for simplicity.

Figure 3 shows our results. In general, the operations *Add* and *Receive* display similar trends. *Peek* is the fastest operation, since it does not need to alter the visibility of the message or alter the queue state. *Add* and *Receive* require some sort of synchronization

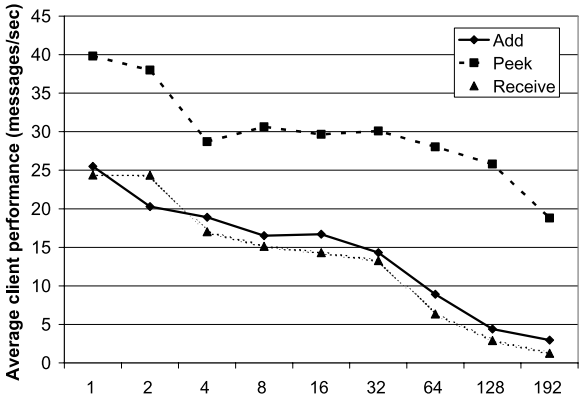

Fig. 3. Average Queue client performance as a function of the number of concurrent clients. Message size is 512 bytes.

to alter the queue state: the *Add* operation needs to add the message to the same places in each replica of the queue; the *Receive* operation needs to assign the triple-replicated message to only one client. For the *Add* and *Receive* operations, the maximum service-side throughput peaks at 64 concurrent clients with 569 and 424 ops/s, respectively. For *Peek*, however, we show the throughput that is achieved with 192 concurrent clients, but the data suggests that we have not exercised this operation fully because service-side throughput is still increasing from 128 to 192 instances – 3878 ops/s for 192 clients compared to 3392 ops/s for 128 clients. Limitations on the number of virtual machines that we can start on Windows Azure prevented us from running this experiment at a higher scale. We have also run some experiments to test the influence of the size of the queue on the performance of the mentioned operations, and found that there is not much variation in performance as the queue grows in size from 200,000 messages to 2 million messages.

4. Azure computing services

In this section, we discuss computing instance acquisition and release time in Windows Azure and TCP communication between different virtual machines instances.

4.1. Dynamic scalability

Computing instance acquisition time is a critical metric to evaluate the efficiency of dynamic scalability for cloud applications. We have written a test program that uses the Windows Azure management API to collect timing information about each possible action on Azure virtual machine instances. We manage both types of VMs: web roles and worker roles. In addition, Azure offers four types of VM size: small, medium, large and extra large. By combining these two parameters for each test case we create a new Azure cloud deployment.

For every run of our test program, the test program randomly picks a role type and a VM size, and creates a new deployment. In our test, all the deployment packages are stored in Azure blob storage services. We choose the number of instances in each deployment based on the VM size in order to stay below the 20-core limit imposed by Azure on normal user accounts and still allowing the deployment size to double: 4 instances for small, 2 for medium and one for large and extra large. Then our test program measures the time spent in all five phases – create, run, add, suspend and delete. These phases are divided based on Azure deployment and instance status:

(1) Create: we record the wall clock time from application deploy request initiation to the time when Azure indicates the deployment is ready to use.
(2) Run: the test program initiates a "Run" request to start the VM instances in the deployment. We measure the time from the start of the request to the time when all VM instances are ready to use (the status goes from "stopped" to "ready").
(3) Add: the test program initiates a "Change" request and doubles the number of running instances. We measure the time that takes these newly added instances to become ready, which is indicated by a status change to "ready".
(4) Suspend: we suspend all the running instances in the deployment and measure the time spent to terminate each Azure VM instance (status changes from "ready" to "stopped").
(5) Delete: After all the instances are suspended, initiate a "Delete" request and removes the current deployment.

From December 17, 2009 to January 09, 2010, we collected data from 431 successful runs. The VM startup failure rate, taking into account all of our test cases, is 2.6%. Starting January 1, 2010, Windows Azure changed from CTP to commercial platform, and our observations did not find significant performance differences between these two periods. The output data is shown in Table 1. From these tables we draw the following observations:

Table 1
Worker role and web role VM request time (s)

Role	Size	Statistic	Create	Run	Add	Suspend	Delete
Worker	Small	AVG	86	533	1026	40	6
		STD	27	36	355	30	5
	Medium	AVG	61	591	740	37	5
		STD	10	42	176	12	3
	Large	AVG	54	660	774	35	6
		STD	11	91	137	8	6
	Extra large	AVG	51	790	N/A	42	6
		STD	9	30	N/A	19	5
Web	Small	AVG	86	594	1132	86	6
		STD	17	32	478	14	2
	Medium	AVG	61	637	789	92	6
		STD	10	77	181	17	6
	Large	AVG	52	679	670	94	5
		STD	9	40	155	14	3
	Extra large	AVG	55	827	N/A	96	6
		STD	16	40	N/A	3	8

(1) Web role VM instances need longer time to startup than worker role instances. For all VM sizes, web role takes 20–60 s longer than Work role VMs. Also, large VMs take longer time to startup than small VMs.
(2) The average time to start a worker role small instance is around 9 min, while the average time to start a web role instance is around 10 min. The quickest time we observe is 7.5 min for worker role and 9 min for web role. For 85% of our test runs, the first small worker role instance becomes ready within 9 min, and for 95% the first small worker role instance becomes ready within 10 min. For 80%, the first small web role instance becomes ready within 10 min and for 90%, the first small web role instance becomes ready within 11 min.
(3) Azure does not serve a request for multiple VMs at the same time. That is, there is a lag between the time the first instance becomes available and the following ones. For both worker role and web role small instances, we have observed a 4 min lag between the 1st instance and the 4th instance of our deployment.
(4) Adding more instances to existing deployment takes much longer than requesting the same number of instances initially.
(5) Application deployment performance – create phase – is largely a function of the application size. A 1.2 MB application starts 30 s faster than a 5 MB application. Note that our test deployment is stored in Azure storage service. If the application package is stored locally, the deployment time could takes much longer because of the local network bandwidth limit.
(6) Azure shows consistent performance for deployment deletion, around 6 s for all test cases.

4.2. TCP communication

Windows Azure allows the programmer to define TCP or HTTP internal endpoints for the virtual machine instances in the deployment. This type of communication is highly-coupled, works only in a point to point fashion, and the application needs to define the protocol. However, it is a good complement to the Queue service since these internal ports allow the VMs to talk directly with each other using a low-latency, high-bandwidth TCP/IP port. Therefore, we have measured the performance of this feature of Azure VMs based on latency and bandwidth.

For this experiment, we create a deployment with 20 small VMs. Ten of these VMs measure latency, and the rest measure bandwidth. Each virtual machine is paired with another one; each pair contains one server and one client. In order to measure the latency, the client measures the roundtrip time of 1 byte of information sent on the TCP channel, after communication has been established. For the bandwidth measurement the client sends 2 GB of information to the server – each run of this bandwidth test usually takes around 30 s. Figures 4 and 5 present our results. We have collected for

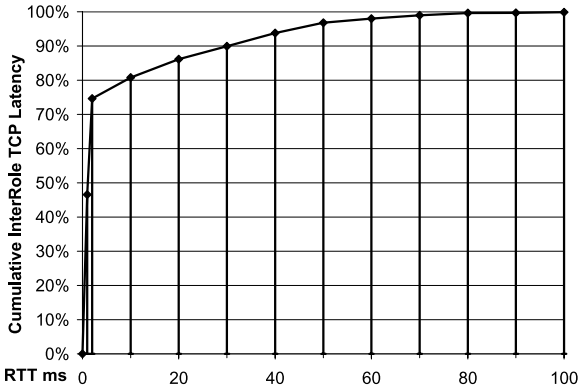

Fig. 4. Cumulative TCP latency between two small VMs communicating through TCP internal endpoints.

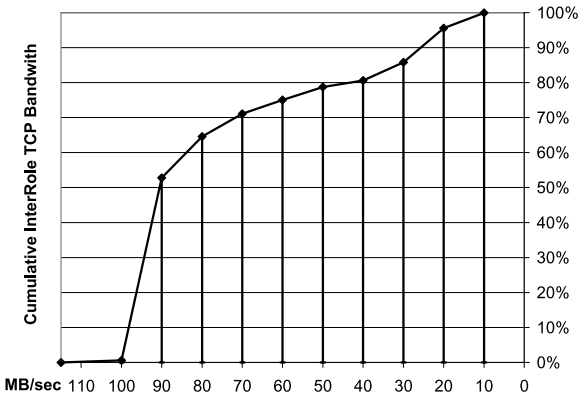

Fig. 5. Cumulative TCP bandwidth between two small VMs sending 2 GB of data through TCP internal endpoints.

these two graphs a total of 10,000 measurements. Both figures present the histogram of our samples. Figure 4 shows that approximately 50% of the time the latency is equal to 1 ms; 75% of the time the latency is 2 ms or better. In general, the most common case is to find in the datacenter latency that is similar to our LAN. Figure 5 summarizes the bandwidth measurements. 50% of the time we find the bandwidth to be 90 MB/s or better. We assume that the physical hardware is Gigabit Ethernet, which has a limit of 125 MB/s. So in our experiments we have seen that is rather common for the VMs to have good bandwidth. However, for the lower end of the sample – 15% – the performance drops to 30 MB/s or worse.

5. Experience with a real eScience application in Windows Azure: ModisAzure

In this section we present our experiences with regard to the performance of Windows Azure for an eScience application, ModisAzure, since it was deployed in February 2010. We will first briefly describe the application and its architecture and then introduce our analysis of its performance on Windows Azure with attention specifically on what we call "VM task execution timeouts". Our description of the application itself is very brief and interested readers are encouraged to examine our other publications for more detail on the development and design of ModisAzure itself [15,16].

5.1. Overview of ModisAzure

ModisAzure is a web-based application to integrate data from ground-based sensors with the Moderate Resolution Imaging Spectroradiometer (MODIS) [14, 19] satellite data and to allow scientists to execute their own analysis code on the integrated data. The MODIS data, generated by the *Terra* and *Aqua* satellites, is designed to improve the understanding of global dynamics and processes occurring on the land, oceans, and lower atmosphere. The dataset is a set of images covering the entire Earth's surface in 36 spectral bands, at multiple spatial resolutions, generated every 1–2 days. The raw data itself is available via FTP, and the size of the data for 10 years of the entire continental United States is approximately 4 TB spread across 585 K input source files.

The ModisAzure web-application is architected as a pipeline framework as seen in Fig. 6. Processing is divided into three stages: data collection, reprojection, and analysis/reduction. A user enters a processing request using the web portal and as part of that request specifies the desired data products to be computed. The request is stored in an Azure table and the set of independent tasks to be completed is determined based on the specific request. Each task describes a specific region of the final data product to be computed and tasks are executed in parallel and do not communicate. The stages are data dependent, however, and thus collection must precede reprojection, and reprojection must precede reduction. Results are saved along the way for reuse later so that work is not duplicated more than necessary.

A brief walk-through of how a request is handled gives a good overview of how the application phases interact. When a user enters the web portal he specifies a set of data products to process as a combination of geographic information (specifically defined regions) as well as a time-span. The request is then added to a service queue which is monitored by a service

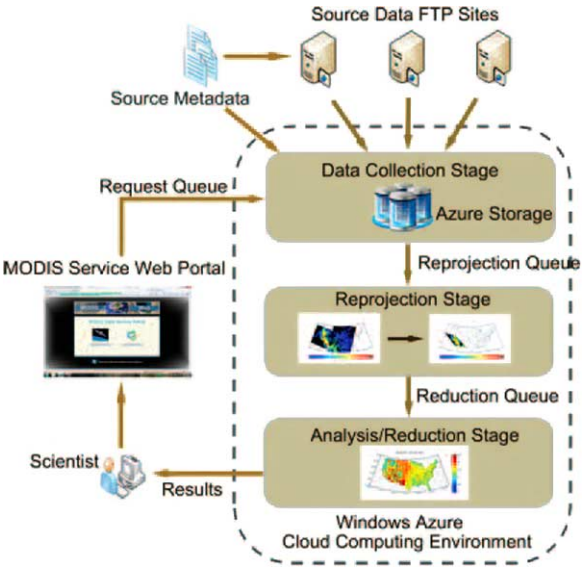

Fig. 6. Pipeline stage view of ModisAzure from [16]. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-2011-0323.)

manager which manages the execution of the requests and their associated tasks. The service manager processes incoming requests and computes how the request is broken into smaller pieces which are handled independently by the various worker role instances (the current deployment uses up to 200 instances concurrently). Each independent piece of work, or task, follows the three principle phases of the pipeline: data collection, reprojection, and reduction. Worker role instances watch queues to get new tasks to work on and as soon as they finish one, they retrieve the next. A single request may generate hundreds or thousands of tasks for a single final data product.

Data collection phase: As described in [16], for each task on a worker role, the data collection stage gathers the required source data based on the specific request received from the user. This may require downloading data files from the satellite data feeds via FTP. The worker instance checks whether the source data has already been downloaded and if not it downloads the data files and stores them in Azure blob storage. If the source files already exist in blob storage then they are retrieved to the local instance storage for the next phase, reprojection. When all source data is available in local instance storage the reprojection phase is initiated on the instance. A typical task requires 3–4 source data files, each of which is typically between several megabytes and tens of megabytes in size.

Reprojection phase: Reprojection is the actual merging and transforming of the data from multiple sources into a single data sub-product (think of a tile in an image mosaic). When a reprojection task is begun the first action is to check to see if this product has been computed and stored previously; and, if it is found then the task is aborted and the stored result is used instead. A single reprojection task typically takes several minutes of computation on a small-size instance in Azure. Once reprojection is complete the finished data product is stored in blob storage for potential reuse later. The last stage, reduction, is then triggered.

Reduction phase: The reduction phase is optional, and works on the data produced by the reprojection task. This phase allows scientists to run their own code, in the form of a MATLAB executable or some other command-line executable on the data for a completely custom computation phase. Each reduction task may take several input files and aggregate the data into some set of image files. Upon completion of the reduction phase for all tasks of the request, an email is sent to the user to indicate that the final product data files are available for download from the data repository.

5.2. VM behavior observations in ModisAzure

ModisAzure has been in operation since February, 2010, and nearly 3 million distinct tasks were executed between February, 2010 and September, 2010, the time period of log data over which we present our analysis. Table 2 shows the breakdown of tasks executed for each phase and type. The number of tasks executed is not the same as the number of distinct tasks because in the case of a failure a task is retried, so a single task may account for multiple execution runs in order to complete successfully. The table shows a type of task, aggregation, that we have not previously discussed, but it is simply a precursor task to a reduction task that groups data together before an additional reduction step is performed as a Reduction-type task. Reprojection and Reduction are the two dominant phases due to the reuse of both source downloads and aggregation products. In Windows Azure the cost to store 1 GB for 1 month is nearly the same as it does to run a small VM instance for one hour so storing intermediate products to conserve computation is a valid strategy as long as the data is used within a month.

The system experienced task failure for a variety of reasons, including user-code errors (e.g., bugs in user-supplied MATLAB code), download failures, VM startup failures and blob retrieval errors, but here we focus on one particular class of failure which we found to be both interesting and difficult to diagnose or pre-

Table 2
ModisAzure task breakdown and selected failure types

ModisAzure task classification	Task execution count	Percentage of total (%)
Source download	139,609	4.57
Aggregation	8706	0.29
Reprojection	1,704,002	55.79
Reduction	1,202,113	39.36
Total task executions	3,054,430	100.00
Selected types of task errors		
Success	2,000,656	65.50
Unknown failure	345,180	11.30
Blob already exists	182,726	5.98
Unknown – null log	139,609	4.57
Download source data failed	125,164	4.10
Connection failure	8966	0.29
VM execution timeout	5300	0.17
Operation timeout	4178	0.14
Corrupt blob read	3107	0.10
Server busy	1287	0.04
Blob read fail	638	0.02
Non-existent source blob	519	0.02
Unable to read input file	20	0.00
Bad image format	15	0.00
Transport error	12	0.00
Internal storage client error	10	0.00
Out of disk space	7	0.00

dict: VM task execution timeout. We found that in some instances a task would seemingly execute normally, not fail explicitly, but would be much slower than other similar tasks. We tested explicitly for this condition by monitoring each task's execution and if it was still executing after 4× of the average completion time for that task it would be cancelled and retried. The VM timeout was triggered in 5300 task executions over the studied time period of February to September in 2010 representing 0.17% of all task executions. While that is a small percentage, if left unchecked such performance problems can significantly hinder performance for a given request if even one task experiences the slowdown. For example, in ModisAzure the timeout was triggered after a single task executed for more than 45–60 min, depending on the specific task, but a normal task execution completed within 10 min.

Figure 7 shows the daily percent of task timeout failures out of all daily task executions over time. The percentage of tasks that experienced timeout ranges from 0% to nearly 16% where the percentage is from the number of timed-out tasks divided by the total number of tasks executed that day. This shows the variable nature of the cloud infrastructure and even though the VM execution timeouts are a small fraction of the overall task count, on any given day they may cause an application to take up-to 48% longer to execute ($16\% \times 4 + 84\% = 148\%$) if 16% of the tasks timeout after running for 4× their average historical execution time. Thus, this is an error that is worth monitoring for and a good task execution history may allow even tighter bounds than the 4–5× we used in order to minimize wasted time and hence cost as well.

We have already discussed the impact of VM Execution Timeouts on application performance, but there are other failure types worth examining as well, shown in Table 2. Connection failures, corrupt blob reads and blob read failures are all failures that while not common, must also be accounted for in large scale cloud applications. We have omitted many other failure types which were primarily related to user-provided MATLAB code, and hence the table does not represent 100% of all task executions. Due to space constraints, we will not discuss all of these error types and their causes. We believe this data is useful for eScience developers because it shows the kinds of errors that can

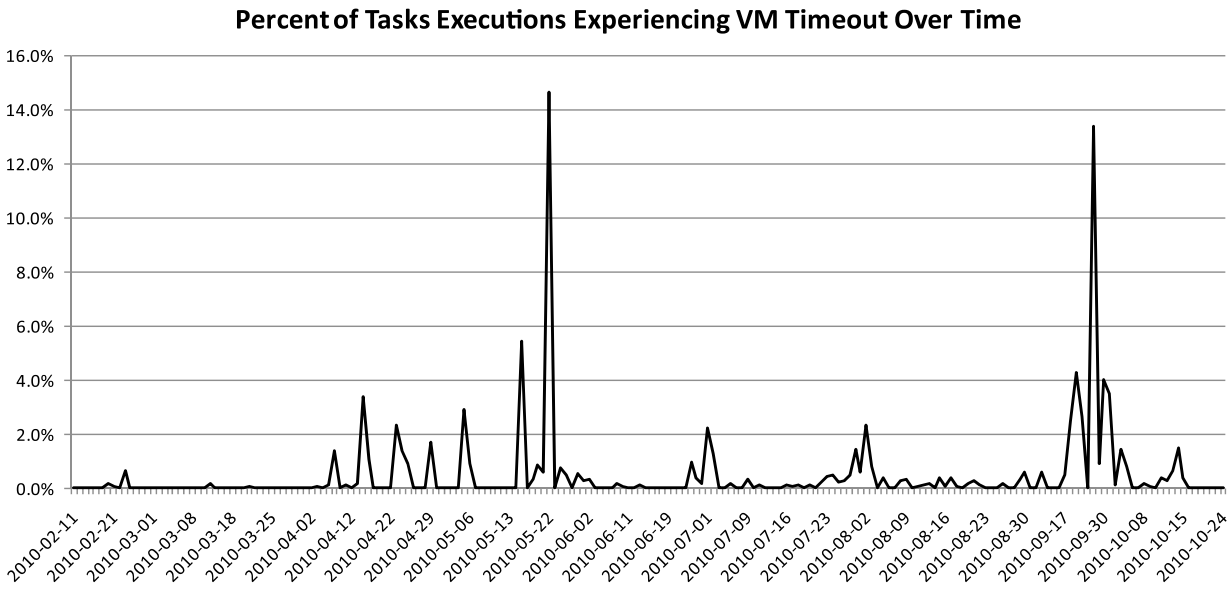

Fig. 7. Percent of task executions with VM timeout over time.

occur in clouds at scale and that at scale errors which only occur $<1\%$ of the time can still occur many thousands of times.

To address the failures and performance issues we encountered in ModisAzure we implemented robust task status tracking and retry mechanisms. Initially we relied upon queue storage's retry mechanism in which a queue message that is not explicitly removed after a specified time-period will re-appear in the queue automatically. However, we found this to be insufficient in cases where tasks take longer than the maximum visibility timeout value (2 h) as well as for handling cases where the a task is being executed slowly and allowing another worker to execute the same task concurrently could cause corrupted output. Therefore, we used explicit task status monitoring by a task manager worker that was responsible for checking for task timeouts and killing slow tasks and putting the task back into the task queue to be re-run by another worker.

6. Recommendations

In this section, we present our recommendation for developers and users of the Windows Azure cloud. These recommendations are based upon our experimental results and our experience developing scientific applications for Windows Azure.

6.1. Azure storage services

We recommend using some extra data caching mechanisms on the client-side to expand the per-client bandwidth limit, and using data replication on the blob storage to expand the server-side bandwidth limit. The blob storage download bandwidth, when accessed from small instance types, is limited by the client's bandwidth for small numbers of concurrent clients. For 1–8 concurrent clients we saw a 100 Mbit/s, or approximately 13 MB/s, limitation. We have observed a per-client bandwidth drop of approx 1.5 MB/s when we doubled the number of concurrent clients. The maximum service-side bandwidth achievable against a single blob for a high number of concurrent clients is limited to approximately 400 MB/s, which is just about what we would expect from three 1 GB/s links if a blob is triple-replicated.

In order to get the best performance out of the table service, the table entities should be accessed by using partition keys and row keys only. Particularly, users should avoid querying tables using property filters under performance-critical or large concurrency circumstances. Currently, all tables are indexed on the *PartitionKey* and *RowKey* of each entity, and creating an index on any other properties cannot be specified. Performance suffers when querying on non-indexed properties. In one of our experiments, over a half of the 32 concurrent clients got time-out exceptions instead of correct results when querying the same table par-

tition – with ∼220,000 entities pre-populated – using property filters.

Multiple queues should be used for supporting many concurrent readers/writers. We found that performance degraded as concurrent readers and/or writers were added, but each client obtained on average more than 10 operations per second for message sizes of 512 bytes to 8 kB for up to 32 writers. With 16 or fewer writers each client obtained 15–20 ops/s. We also found that message retrieval was more affected by concurrency than message put operations so users cannot assume similar scale at each end of the queue.

6.2. Dynamic scaling

If fast scaling out is important, hot-standbys may be required if a 10 min delay is not acceptable, although this option would incur a higher economic cost. Dynamically adding VMs to a deployment at runtime is a useful feature of Azure enabling dynamic load matching, but you should be aware that it often takes on the order of 10 min from the time of the request until the instance is integrated into the deployment. Additionally, web roles took, on average, 60 s longer to come up for small instance types, and about 30 s longer for medium to extra large instance sizes.

6.3. Application development and deployment

Build a robust logging and monitoring infrastructure early in the project. During the development and deployment of ModisAzure we learned several important lessons. Because of the variable and black-box nature of the platform, extensive monitoring and logging facilities are necessary to not only diagnose problems but also to determine how the application is behaving. Windows Azure presents particular difficulties because of its platform-as-a-service nature, which prohibits operating-system level access to the virtual machine as well as the non-durable nature of the local VM instance storage. Building a robust logging and monitoring infrastructure early in the project is advantageous because errors that did not occur at lower scale will begin to become common as scale increases. This is further enhanced by the ease and speed at which scaling can occur for a cloud application as opposed to a traditionally hosted application that would require additional hardware purchases for scaling and thus give developers more time to prepare for the problems inherent with high-scale.

7. Conclusion

In this paper we have presented the results from experiments we have conducted on Windows Azure. We have shown an exhaustive performance evaluation of each of the integral parts of the platform: virtual machines and storage services as well as shown the challenges in both performance and reliability faced by a real eScience application. Based on these experiments, we also provide our performance-related recommendations for users of the Windows Azure platform. These cloud services are the building blocks for cloud applications, and are usually presented to the user as a black-box, with no performance guarantees. Our main focus is to provide the community with performance information and concrete recommendations that help the design and development of scalable cloud applications.

Acknowledgements

We wish to thank Catharine van Ingen of Microsoft Research for her collaboration and assistance during the development of ModisAzure and the subsequent deployment and data analysis.

References

[1] Amazon, Amazon Elastic Compute Cloud (EC2), available at: http://aws.amazon.com/ec2/.

[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin and I. Stoica, Above the clouds: a Berkeley view of cloud computing, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, CA, 2009.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, Xen and the art of virtualization, in: *Proc. 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, USA, October 19–22, 2003, ACM, New York, NY, pp. 164–177.

[4] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg and I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, *Future Generation Computer Systems* **25**(6) (2009), 599–616.

[5] E. Deelman, G. Singh, M. Livny, B. Berriman and J. Good, The cost of doing science on the cloud: the montage example, in: *Proc. 2008 ACM/IEEE Conference on Supercomputing*, IEEE Press, Piscataway, NJ, 2008, pp. 1–12.

[6] J. Dejun, G. Pierre and C.H. Chi, EC2 Performance analysis for resource provisioning of service-oriented applications, in: *Proc. 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing*, Stockholm, Sweden, November 23, 2009.

[7] C. Evangelinos and C.N. Hill, Cloud computing for parallel scientific hpc applications: feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2, in: *Proc. Cloud Computing and Its Applications 2008*, Chicago, IL, October 2008.

[8] I. Foster, Y. Zhao, I. Raicu and S. Lu, Cloud computing and grid computing 360-degree compared, in: *Proc. Grid Computing Environments Workshop (GCE'08)*, IEEE Press, Austin, TX, 2008, pp. 1–10.

[9] S.L. Garfinkel, An evaluation of Amazon's grid computing services: EC2, S3 and SQS, Technical report, Center for Research on Computation and Society School for Engineering and Applied Sciences, Harvard University, 2007.

[10] Google, Google App Engine, available at: http://code.google.com/appengine/.

[11] Z. Hill, J. Lie, M. Mao, A. Ruiz-Alvarez and M. Humphrey, Early observations on the performance of Windows Azure, in: *Proc. 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*, Chicago, IL, 2010.

[12] Z. Hill and M. Humphrey, A quantitative analysis of high performance computing with Amazon's EC2 infrastructure, in: *Proc. 10th IEEE/ACM International Conference on Grid Computing (Grid'2009)*, Banff, Canada, October 13–15, 2009.

[13] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman and J. Good, On the use of cloud computing for scientific workflows, in: *Proc. 4th IEEE Int. Conference on eScience (ESCIENCE'08)*, IEEE Computer Society, Washington, DC, 2008, pp. 640–645.

[14] C. Justice et al., The moderate resolution imaging spectroradiometer (MODIS): land remote sensing for global change research, *IEEE Transactions on Geoscience and Remote Sensing* **36**(4) (1998), 1313–1323.

[15] J. Li, M. Humphrey, Y.-W. Cheah, Y. Ryu, D. Agarwal, K. Jackson and C. van Ingen, Fault tolerance and scaling in e-Science cloud applications: observations from the continuing development of MODISAzure, in: *IEEE e-Science 2010 Conference*, Brisbane, Australia, December 7–10, 2010.

[16] J. Li, D. Agarwal, M. Humphrey, C. van Ingen, K. Jackson and Y. Ryu, eScience in the cloud: a MODIS satellite data reprojection and reduction pipeline in the Windows Azure platform, in: *Proc. 24th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS 2010)*, Atlanta, GA, April 19–23, 2010.

[17] A. Menon, J.R. Santos, Y. Turner, G. Janakiraman and W. Zwaenepoel, Diagnosing performance overheads in the Xen virtual machine environment, in: *Proc. 1st ACM/USENIX Int. Conference on Virtual Execution Environments*, Chicago, IL, June 11–12, 2005, ACM, New York, NY, pp. 13–23.

[18] Microsoft, Windows Azure Platform, available at: http://www.microsoft.com/azure/default.mspx.

[19] NASA MODIS Web page, http://modis.gsfc.nasa.gov.

[20] M.R. Palankar, A. Iamnitchi, M. Ripeanu and S. Garfinkel, Amazon S3 for science grids: a viable solution?, in: *Proc. Int. Workshop on Data-Aware Distributed Computing*, Boston, MA, June 24, 2008, ACM, New York, NY, pp. 55–64.

[21] A. Ranadive, M. Kesavan, A. Gavrilovska and K. Schwan, Performance implications of virtualizing multicore cluster machines, in: *Proc. 2nd Workshop on System-Level Virtualization For High Performance Computing*, Glasgow, Scotland, March 31, 2008, ACM, New York, NY, pp. 1–8.

[22] E. Walker, Benchmarking Amazon EC2 for high-performance scientific computing, *Login The Usenix Magazine* **33**(5) (2008), 18–23.

[23] L. Youseff, R. Wolski, B. Gorda and C. Krintz, Evaluating the performance impact of Xen on MPI and process execution in HPC systems, in: *Proc. 2nd Int. Workshop on Virtualization Technology in Distributed Computing*, IEEE Computer Society, Washington, DC, 2006, p. 1.