Preliminary version. Final version appears in *Proceedings of 7th IEEE International Conference on e-Science (escience 2011).* Stockholm, Sweden, Dec 5-8 2011.

Assessing the Value of Cloudbursting: A Case Study of Satellite Image Processing on Windows Azure

Marty Humphrey, Zach Hill
Department of Computer Science
University of Virginia
Charlottesville, VA 22904

Keith Jackson
Lawrence Berkeley National Lab
Berkeley, CA

Catharine van Ingen
Microsoft Research, Bay Area Research Center
San Francisco, CA

Youngryel Ryu
Seoul National University
Seoul, Korea

Abstract— **To perform computational experiments at greater scale and in less time, enterprises are increasingly looking to dynamically expand their computing capabilities through the temporary addition of cloud resources (aka "cloudbursting"). Computational infrastructure can be dismantled in minutes with no long-term capital investments. However, research is needed to identify *which* properties of an application best determine the potential benefits of cloudbursting. For example, there are certainly situations where the cost to transfer the necessary input data from the enterprise to the cloud (to execute the application in the cloud) outweighs the value of simply waiting until resources become available in-house. To better understand and quantify these general issues, we perform a concrete analysis of the value of cloudbursting for a large-scale application we have previously created to process and derive environmental results from satellite imagery. More specifically, we compare three versions of the application (an all-cloud design; a version that runs in-house on our cluster; and a hybrid cloudbursting version) on dimensions of debuggability, fault tolerance, correctness, economics, usability, and run-time speed. We find that for our application, cloudbursting is effective primarily because we were able to design the application so that its I/O behavior does not preclude remote (cloud) execution, we were able to minimize developmental cost by constructing a cloud run-time environment that is very similar our in-house environment, and we achieve good run-time performance in our cloud-based executions (for example, we describe how a representative computation that takes 2 ½ hours in-house is completed in 35 minutes via cloudbursting). By generalizing this analysis, we believe that we contribute guidance to the broader community on the value of cloudbursting for escience applications.**

Keywords-cloud computing, Windows Azure; cloudbursting; MODIS.

I. INTRODUCTION

For many enterprises, the appeal of cloud computing is *not* to wholly replace in-house computational infrastructure with cloud resources, but rather to selectively and opportunistically *augment* in-house resources with cloud resources. This is the "best of both worlds", whereby dynamic scalability is achieved without large capital expenditures, and without the upheaval of porting/rewriting applications to run solely in a cloud environment. This "cloudbursting" methodology allows maximum productivity from existing in-house resources, while still facilitating a future environment where cloud resources are increasingly relied upon for day-to-day operations, including escience experiments.

However, not every application can execute effectively in a cloudbursting model. The extreme cases are understood – for example, a tightly-coupled MPI application is unlikely to be successful if cloudbursted in the wide-area across enterprise and cloud resources, as network latency becomes intolerable. On the other hand, a bag-of-tasks Monte Carlo simulation with minimal I/O might be an excellent candidate for cloudbursting. In between, however, there are applications that might significantly benefit (or not) from cloudbursting, and the dominant reason might be different for each. One application might work because it can be easily ported to leverage in-cloud storage capabilities (similarly, an application might work because no such porting is even necessary!) Another application might be a good fit for cloudbursting because the economics of running it in the cloud are particularly appealing. A third application might not make sense for cloudbursting because the input data for the application is too big to justify moving from the enterprise to the cloud just for the execution (and of course some enterprises might decide that there are security requirements that prevent the application from ever running in a cloud. Research is needed to identify *which* properties of an application best determine the potential benefits of cloudbursting.

To better understand and quantify these general issues, we perform a concrete analysis of the value of cloudbursting for a large-scale application we have previously created to process and derive environmental results from satellite imagery. Our application, MODISAzure [1][2], is one of the first escience applications to use the Microsoft Windows Azure cloud platform. A typical execution of MODISAzure produces an analysis of environmental characteristics for each day being studied (in a bag-of-tasks style) and then aggregates the day-of-year results (more details are contained in Section 3 of this paper). To study the value of cloudbursting, we created and compare three versions of this application. The first version executes entirely in Windows Azure. The second version is a port of the Windows Azure version to run entirely on our

Windows HPC cluster. The main modifications that were necessary were to replace the use of cloud storage (e.g., blobs, tables, queues) and to replace our custom job-management software in Windows Azure with invocations to the Windows HPC scheduler. Our third version is the cloudbursting version, capable of running in-house on our Windows HPC system as well as inside the Windows Azure cloud. As we describe in the paper, this third version was not simply a sum of the first two versions, as this hybrid cloudbursting model required modifications to the first two versions in order to be effective.

We find that for our application, cloudbursting is effective primarily because we were able to design the application so that its I/O behavior does not preclude remote (cloud) execution, we were able to minimize software development cost by constructing a cloud run-time environment that is very similar our in-house environment, and we achieve good run-time performance in our cloud-based executions (for example, we describe how a representative computation that takes 2 ½ hours in-house is completed in 35 minutes via cloudbursting). By generalizing this analysis on dimensions of debuggability, fault tolerance, correctness, economics, usability, and run-time speed, we believe that we contribute guidance to the broader community on the value of cloudbursting for escience applications.

The rest of the paper is organized as follows. Section II contains the related work. Section III describes the science we are pursuing involving the MODIS satellites. Section IV describes the three versions of the application. Section V contains the evaluation and discussion. Section VI concludes.

II. RELATED WORK

There are a number of emerging "science clouds" [3] [4][5]. Similarly, there is an increasing number of science applications in otherwise non-science clouds (e.g., CloudBLAST [6], coupled atmospheric-ocean climate models [7], data mining [8], astrophysics [9], astronomy [10][11], high energy physics [12]). Windows Azure has been used as the platform for executing BLAST very effectively [13][14]. Whereas the focus of these projects is to get an application to execute entirely in a cloud, we attempt to create an implementation that can run effectively within the enterprise, in the cloud, or both.

Beyond facing the initial challenges of just getting an application to run in a cloud at all, there is an increasing recognition of performance concerns of clouds and their underlying technologies. One of the first studies was provided by Garfinkel [15], who evaluates some of the cloud services that Amazon provides. Similarly, local HPC clusters were compared against EC2 [16]. Another report examines the feasibility of using EC2 for HPC in comparison to clusters at NCSA [17], pitting EC2 against high-end clusters utilizing Infiniband interconnects. A recent study considered the newest support for MPI in EC2 against local clusters [18].

The scheduling of bag-of-tasks applications in a cloudbursting model is the focus of a recent study [19], in which it is assumed that the application can be executed in the cloud, and the issue is thus how to schedule it so that its overall time to completion and cost is minimized. In contrast, in our work, we investigate whether or not the application is suitable for this cloudbursting execution at all.

III. BACKGROUND: MODIS AND WINDOWS AZURE

We first describe the nature of the scientific computation we are performing. We then give an overview of the Microsoft cloud: Windows Azure.

A. *MODIS and the Scientific Computation*

The MODIS (Moderate Resolution Imaging Spectroradiometer) sensor, on board the Terra and Aqua satellites, was designed to improve the understanding of global dynamics and processes occurring on the land, oceans, and atmosphere [20][21]. The MODIS data is a view of the entire Earth's surface in 36 spectral bands, at multiple spatial resolutions, generated every 1-2 days. There are a large number of research activities that currently use the MODIS data to explore and validate scientific hypotheses (e.g., see [22] for an overview with regard to vegetation and [23] for an overview with regard to ocean science).

The science goal of our project is to apply the MODIS data in the calculation of the evapotranspiration (ET) on the earth surface. ET controls land-atmosphere feedback and is an important source of water vapor to the atmosphere. Atmospheric water vapor is the most significant greenhouse gas and key to understanding hydrologic cycle. As such, ET plays a fundamental role in weather and climate. Our computation of ET involves multiple science data products from the MODIS source data sets, each containing a specific type of earth surface imagery (such as Land surface temperature or atmospheric aerosol). Each data product contains a number of related science variables separated into HDF format source files. The size of a single data product for each global year ranges from several hundred GBs to over 1 TB. Although files can be downloaded from the FTP sites, currently there are no public software frameworks available in the earth science community to automate the processes of reconciling and cataloging the various data products and/or scheduling parallel computations over these data. As a result, scientists need to handle processing complexities manually, which often prevents large scale analyses.

There are two computations we are performing. The first is *reprojection*, in which the heterogeneous data products collected from the appropriate NASA FTP sites are reprojected into time and spatial-aligned imagery data. A set of compute-intensive algorithms (e.g. nearest neighbor pixel match) is performed to harmonize data points pixel by pixel. Typically, a single reprojection task requires a collection of only 3-4 source data files, each in the size of several MBs to tens of MBs. The computation required to produce a target sinusoidal file (i.e., reproject the source data into a uniform format) can be finished in 10-15 minutes on a typical commodity PC. The second is *reduction*, in which a scientist performs an analysis computation over the reprojected data from the previous stage. A "first reduction stage" is used for science derivation such as computing a new science variable from a number of input variables. The "second reduction stage" is used for subsequent analysis of spatial or temporal aggregates. The first reduction stage does the science computation at scale; the second reduction stage creates the smaller science analysis artifacts necessary to understand the results of the first stage.

B. Windows Azure

Windows Azure was announced by Microsoft as its cloud computing platform at its Professional Developers Conference (PDC) Nov 2008. Windows Azure presents a .NET-based hosting platform that is integrated into a virtual machine abstraction. Thus, developers who are familiar with .NET application development can take advantage of this homogeneous cloud environment and develop applications for Azure just like ordinary .NET applications by using Visual Studio. In contrast to this "Platform as a Service" (PaaS), Amazon's EC2 has focused on support for virtual machine technology (aka Infrastructure as a Service, or IaaS). Microsoft has recently augmented its PaaS with IaaS as well. In both EC2 and Windows Azure, users can customize the environment for their application by installing specific software or by purchasing particular machine images.

In Windows Azure, the virtual machine instances can be separated into three different roles: the front-end website hosting server instances are called Web Roles, the back-end computational instances called Worker Roles, and the new Virtual Machine roles. Developers can specify the number of instances for roles at the deployment of their application or can dynamically adjust the number of instances at runtime.

Windows Azure provides three types of cloud storage services, in addition to SQL Azure:

- Blob service, the main storage service for storing durable large data items;
- Queue service, which provides a basic reliable queue model to allow asynchronous task dispatch and to enable service communication;
- Table service, which provides the structured storage in the form of tables and supports simple queries on partitions, row keys, and attributes.

The key aspect of cloud storage is that it is accessible via any virtual machine in Azure (with the proper authentication/authorization). Therefore, while there is local storage available to a particular computation, it is assumed that one of the cloud storage services will be used if the data is to be shared across virtual machine instances.

IV. Our Applications for Processing MODIS Data

In this section, we describe each of the three versions of the application we created to process the MODIS data and derive new scientific results. It is important to note what we did *not* design all three versions at the same time. Initially, we designed an application solely for Windows Azure. It was only later that we decided to investigate a cloudbursting version.

A. MODISAzure (only Windows Azure)

We began the scientific investigation with a few prototyped algorithms, which we wanted to perform at a scale greater than was realistically possible on a typical desktop computer. In other words, we needed a new platform by which to perform analysis of the MODIS data at finer spatial granularity, and we wanted to perform the analysis over more sensed days. More than anything, we were looking for computational capacity, and the Microsoft Windows Azure platform seemed to be a perfect fit for our requirements. However, we realized that there were a number of key challenges in the use of Windows Azure for this application to process the MODIS data:

1. First version: getting the application to run at all was a challenge, as this was a new platform that did not have the familiar mechanisms/abstractions for compute and storage. The latter in particular – queues, blobs, and tables – replaced the traditional distributed file system abstraction.
2. Debugging: we quickly realized that we could not debug our application in our typical way (interactively). That is, Windows Azure is PaaS, so we could not simply start up a remote desktop session (RDP) to the machine to determine what happened. [Note that Windows Azure recently began supporting RDP access, greatly aiding debugging.]
3. Correctness: we decided it would be necessary to add fault-tolerance techniques into our code to handle either unavailable input data or quirky behavior in the cloud platform. For example, we use Terra data and only switch over to Aqua if the Terra data was not available (e.g., because of a satellite outage). However, if a particular instance actually *used* the Aqua data, it was not clear if this was because Terra was truly not available (which would be correct behavior) or our application for some reason failed to find the available Terra data (which would be incorrect behavior). It was a challenge to be able to deal with problems that can arise in the cloud because of scale and *not* mask our own incorrect behavior. Similarly, it was not clear from the first stages which Windows Azure transient faults would masked from us and which issues would rather be seen by our application.
4. Economics: How do we perform the science as cheaply as possible, does our monthly bill match our expectations (e.g., at scale idle machines and unused/forgotten data stored in the cloud could become very expensive), and can we predict how much the next experiment will cost?
5. Performance/scalability: Intuitively, parallelism in our MODIS processing would be achievable through both a space and time decomposition. That is, different regions of the Earth could be processed largely independently, and furthermore different days could be processed independently. However, it was not clear how this implied granularity matched the support for computational parallelism, which was at the virtual machine abstraction. In other words, our only real choice was to instantiate a number of virtual machines, and have them pull 'tasks' from a common queue. It was not clear if this coarse structure would limit our performance and/or scalability.

Figure 1 shows our implementation of the MODISAzure application. The scientist uses a Web portal (not shown) to submit jobs to the system ("Service Request Queue"), where a job can be either to reproject some MODIS data or to perform a reduction. Our Web-facing Windows Azure VM parses the user request into a large number of parallel data collection, reprojection, and reduction tasks (“Reprojection Task Queue” is shown; other queues are similar). During the processing of a user request in the pipeline, the task scheduler also keeps track of the task statuses and retries/requeues tasks as necessary. Each computational worker role instance fetches tasks from the task queue and performs the corresponding data processing work. The nature of a “task” is to: [1] retrieve all the necessary input HDF files from Blob storage; [2] spawn the Matlab process that [a] reads in all of the HDF files, [b] analyzes the data, and [c] writes out its results; and [3] copies the output data back into Blob storage. The particular Matlab job is not memory-intensive. Details can be found in [1][2].

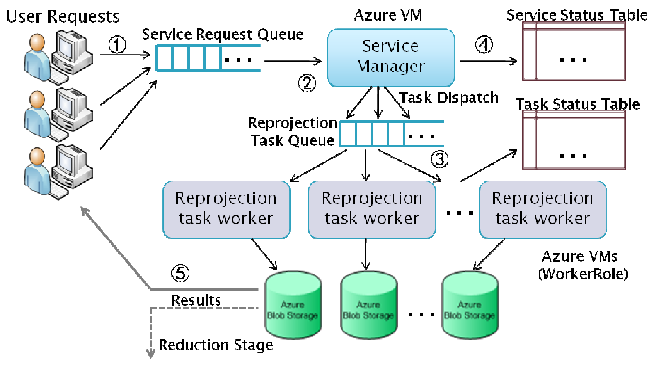

Figure 1. MODISAzure design

Figure 2 shows the aggregate usage of our MODISAzure application. In Spring 2010 we were focusing on the continental US, Summer 2010 was on Global scale reprojection, the rest of 2010 was on Global scale reduction, and the rest of the time was archive download. We have used more than 250,000 CPU hours, amassed an over 5 TB dataset and gained as a factor of over 100x speedup over our reference high-end desktop.

B. *MODISHPC (only Windows HPC)*

We believe that MODISAzure was successful, both because we believe that we performed novel environmental science experiments and also because it gave us a concrete experience with designing, implementing, and debugging a large-scale science application in the cloud. However, there were a number of limiting aspects of the MODISAzure application:

- Debugging was not easy for MODISAzure – we changed our development style because we could not interactively debug our cloud application. That is, we had to replace our typical rapid development cycle with a coarser-grained methodology based essentially on logging statements. This greatly slowed the progress by which we could develop and assess the correctness of our application.

- Scaling (dynamic) took too long for MODISAzure – if we had a relatively large queue and wanted to add capacity to service the requests more quickly, our only option was to add new VMs. The problem is that it took about 15 minutes for this new collection of VMs to become ready to start processing items in the queue. While this is sufficient for many large requests, it was not as quick as we would have liked.

- Economics were not a clear win: while the pay-as-you-go nature was very appealing, and the cost-per-experiment was arguably pretty low, at times we could not escape the feeling that we were wasting money.

- Performance – microbenchmarks indicated that our observed behavior was consistent with that promised by the Windows Azure Service-Level Agreement (SLA), but we still did not know if we could do better without significant effort (via just running on a different platform)

- “Auxiliary Routines” – We had to create a fairly large number of auxiliary routines that were arguably distracting from our scientific pursuits. The most obvious examples of this were that we essentially had to build our own (minimal) queuing system to manage tasks and our own utility for results downloading. The time to write and debug this took away from the direct science experiments.

Because of these concerns, we decided to port the MODISAzure application to run on a Windows HPC cluster. We were already comfortable with the Windows HPC server system, so we were able to accomplish this fairly quickly. Our methodology was [a] change as little as possible in order to get the application to run (we preferred to not have two separate code bases, if possible), and [b] *then* investigate possible extensions to customize the application to exploit the unique characteristics of the HPC environment. As such, there were two primary modifications necessary. First, instead of our custom queue that we created for Windows Azure, we instead used the Windows HPC SDK to submit jobs to the Windows HPC queuing system. This was not difficult, as we were already familiar with programmatically submitting jobs. The second modification was to continue to use the “blob interface” but instead create an implementation of the interface that uses the local file system of the cluster head node and the internal nodes as warranted. This design is consistent with a “run-anywhere” application that we believe is increasingly appropriate for applications and is a simplified version of our longer-term research project to construct a “Cloud Storage Adaptation Layer” (CSAL) [24] that provides an intelligent and adaptive layering between the application logic and the multiple local and non-local storage options.

Recall that in MODISAzure we copied a potentially large number of input HDF files into the working directory of the Matlab executable before it executed. In MODISHPC, we retained this pattern, except that we cache content on each internal node (instead of retrieving from a single folder resident on the head node). This modification – and similarly the use of hard links with the file system – was not trivial, as this introduced issues related to multiple concurrent readers/writers attempting to access and/or update the cache. This was the only “fault tolerance” logic that we found necessary to add (recall that we were starting with our MODISAzure code base, so we already had code for application-level fault tolerance).

The biggest advantage of MODISHPC over MODISAzure was with code development and debugging. Because it was all local, we were able to debug interactively and quickly start/stop jobs. In particular, because we were using a small cluster, we found it very valuable: [a] to be the only user on the cluster (so we could leave and return where we left off, knowing that the

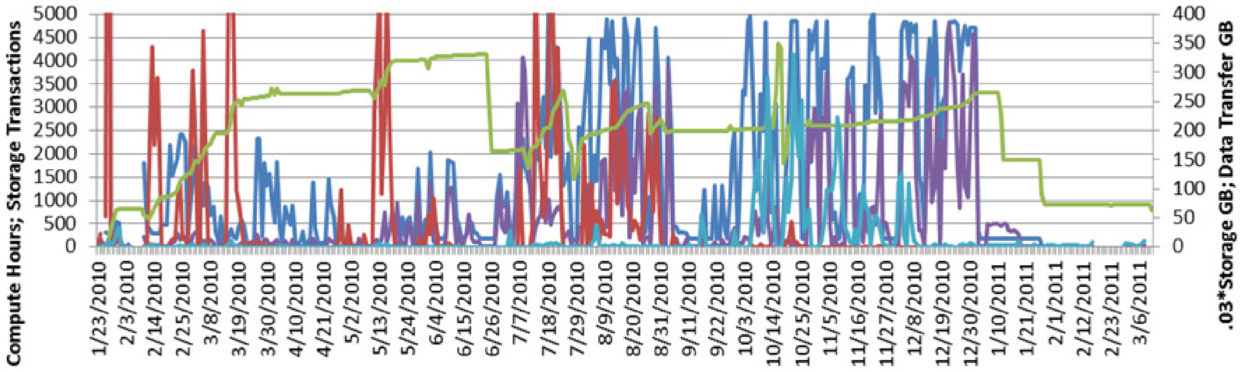

Figure 2. MODISAzure runtime on Windows Azure

state of the system was retained); [b] to have a "debugging folder" on each machine, where each computation on a machine would write error logs; and [c] to monitor the execution of a particular science experiment by real-time viewing of all of the debugging folders via a console on the head node – we watched both the queuing system and the output of the jobs in real-time. When error logs appeared, we "remote desktop" directly onto a node to investigate the error and/or re-run the problem computation by hand to catch the error in real-time and correct the code.

We benefited by replacing our minimal custom queuing system in Windows Azure with the full-featured Windows HPC queuing system, even as we left it to future research to investigate scheduling policies other than essentially FIFO. At the very least, its GUI made it very easy to assess the status of the jobs. Additionally, one aspect of the Windows HPC scheduler that facilitated debugging was that the HPC cluster uses an underlying SQL Server database to keep track of jobs. This proved to be valuable to use because we could use SQL Server Management studio to directly access the "raw data" as necessary to determine what was going on in those situations where we believed that the HPC cluster management console was not providing us directly the information we need. For example, by using SQL Server Management studio to explore the job data, we realized that certain jobs were re-queued automatically when patches were applied to an internal node. Without this capability, we would have believed that the tasks failed for some unknown reason, leaving us to speculate that our code was causing the problematic behavior.

C. *MODISHybrid*

MODISAzure and MODISHPC were successful in general, but each had limitations. For MODISAzure, the development cycle was too long to really facilitate rapid progress. For MODISHPC, for our particular situation, our Windows HPC cluster was fairly small, so we could not easily scale up/down to meet the dynamic requirements of a particular computation. On Dec 1, 2010, Microsoft announced the general availability of Windows HPC Server 2008 R2 SP1, which included support for adding Windows Azure nodes to a local Windows HPC cluster. We saw this as a way to potentially achieve the best of MODISAzure and MODISHPC, namely the development environment of a local machine with the scalability of the cloud.

After a few configuration steps (e.g., recording the security credentials and account information for Windows Azure), the HPC cluster administrator can start/stop a new Windows Azure node as easily as adding a local enterprise node via the HPC Cluster management console. We used a second mechanism called Windows Azure Connect to provide Virtual Private Network (VPN) capability so that the IP address of the Windows Azure nodes appeared to be within our enterprise IP space. This simplified our code, as computations within Windows Azure were able to securely read/write from our HPC cluster head node (e.g., to retrieve the Matlab reduction executable uploaded by the scientist to the head node).

We utilized the new "hpcpack" command to upload our file packages into Windows Azure Blob storage, and we then subsequently issued a "hpcsync" command on each Windows Azure node install these packages from Blob storage to create a file system structure that matched our enterprise nodes in our Windows HPC cluster so that we could more easily code and debug. We had four packages: our MODIS application functionality, the HPC client package (to potentially submit subsequent jobs to our Windows HPC cluster from our Windows Azure nodes), the Matlab runtime environment, and a collection of default input files. Once a Windows Azure node comes on-line, we also had to run some additional commands from the management console to install/configure these package as necessary (e.g., when a Windows Azure node came on-line, the Matlab runtime package was in the file system of the VM but it still needed to have its installer executed). It should be noted that Windows Azure supports the ability to upload an application-customized VM (instead of taking the default VM and configuring it via the 'hpcpack' command), but we found that it was not necessary for our application.

Recall that our original MODISAzure application retrieved content from Windows Azure Blob storage and placed it directly as files into the Windows Azure VM, and that we changed the "blob implementation layer" in MODISHPC to instead directly engage the local file system. In MODISHybrid, the application retrieves input files based on where it was executing – if in the cloud, then retrieve input files from Blob storage, and if in the enterprise retrieve input files from the file system of the head node. (Inside our application, this was essentially the beginnings of an "#IFDEF CLOUD" programming pattern). To facilitate this, we "pre-staged" input files into Blob storage of Windows Azure, and established a "cache" on each VM of input files – a Windows Azure application would first check to see if *another* computation had already retrieved it from Blob storage. We found that this particular I/O enhancement most improved the speed of our Windows Azure-based computations. All day-of-year computations copy their output to Blob storage no matter where they execute. An optional 2nd stage reduction retrieves all of this output data to use as input.

During our development, we found it most productive to typically have four windows open on our desktop. The first window was the web page for submitting a new job. The second window was the Windows Azure management portal – to monitor our VMs via a direct console to Windows Azure. The third window was the Windows HPC management console – to monitor jobs, to facilitate RDP to the Windows Azure nodes, and to directly run commands on selected Windows Azure nodes. Our final window was RDP to one or more Windows Azure nodes for debugging purposes.

Overall, the development productivity was greatly enhanced as compared to the situation in which we could only use Windows Azure. Similarly, the cloudbursting capability was well-integrated with the Windows HPC management console, and we could add/remove nodes easily. The biggest open issue at this point is the degree to which the "#IFDEF CLOUD" pattern should be used. On one hand, it is certainly desirable to customize the behavior to its operating environment. On the other hand, this heterogeneity increases code complexity and complicates debugging.

V. Evaluation and Discussion

Our goal is to assess the three systems based on experiment duration and cost – that is, "how long will the science take?" and "how much will it cost?" To focus the analysis, we attempt a representative moderate-scale computation: the analysis of the region of the earth's surface h28v05 (South Korea and parts of Japan and China) for the entire-year 2003 at a resolution of 1km. Furthermore, this analysis focuses on the reduction phase (after the entire 2003 h28v05 data has been reprojected). Overall, there are 365 independent computations, one for each day of the year. On average, each of these 365 computations requires 217 input files (76 matlab routines – 2.49GB, 141 HDF files – 471 MB), produces 173 temp files (416 MB), and produces 32 output files (5.70 MB). As mentioned in the previous section, if the particular day's computation occurs within the enterprise, the input files are taken from the head node, and if it occurs on Windows Azure then the input files are taken from Blob storage. The temp files and output files are written to local storage (within the VM for Windows Azure, and on a worker node for Windows HPC). After all 365 computations complete, a second-stage reduction (Matlab-based) is executed to determine numerous properties from the viewpoint of the entire year. For example, Figure 3 shows the resulting evapotranspiration calculation for h28v05 for 2003.

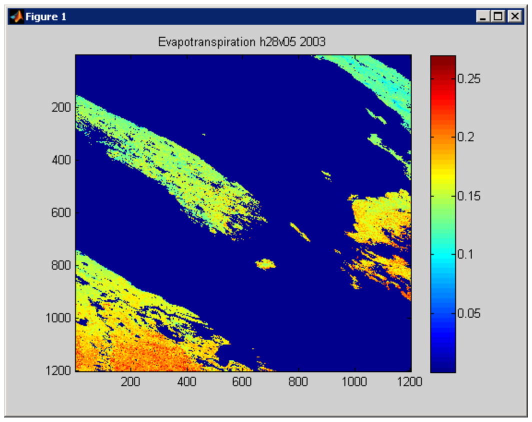

Figure 3. Evapotranspiration (ET) calculated for 2003 h28v05

Part of our research motivation is: do we buy more hardware for our local cluster or do we rent resources from a cloud provider such as Windows Azure? Our cluster was circa-2008, created via commodity hardware: the head node is a Shuttle SN78S dual-core AMD Athlon X2 2.8GHz, with 4GB RAM, with a single hard drive (7200 RPM, 640 GB). There are two internal nodes to the cluster. Each is dual-quad-core (AMD Opteron 2344 HE 1.7GHz), 16GB RAM, one hard drive (C:) 150GB 10K RPM, and another hard drive (D:) 640 GB 7200 RPM. A Netgear 10/100/1000 switch is used.

Given that our cluster was at UVa, there were only two Windows Azure datacenters to realistically choose between: Chicago and San Antonio. We found to the edge of the Chicago datacenter, latency was 21 ms, download bandwidth was 91.1 Mbps, and upload bandwidth was 30.1 Mbps. To the edge of the San Antonio datacenter, latency was 43 ms, download bandwidth was 61.7 Mbps, and upload bandwidth was 15.9 Mbps. We chose to use the Chicago datacenter for our cloudbursting experiments. Typically, it required approximately 45 minutes to fully add new Windows Azure nodes to our enterprise cluster: 15-20 minutes to boot the Windows Azure nodes (in total, irrespective of the number of nodes being added, although a large number generally took a little longer), then install the VPN software and wait for the Windows Azure nodes to register with the VPN, then execute hpcsync, and finally install our software packages.

TABLE I. Windows Azure Instances

Compute Instance Size	CPU	Memory	Instance Storage	I/O Perf	Cost per hour
Extra Small	1.0 GHz	768 MB	20 GB	Low	$0.05
Small	1.6 GHz	1.75 GB	225 GB	Moderate	$0.12
Medium	2 x 1.6 GHz	3.5 GB	490 GB	High	$0.24
Large	4 x 1.6 GHz	7 GB	1,000 GB	High	$0.48
Extra Large	8 x 1.6 GHz	14 GB	2,040 GB	High	$0.96

As is generally the case with other clouds, Windows Azure offers different virtual machines sizes (Table I). Our expected workload for cloudbursting required non-trivial I/O, so we limited our experiments to Medium, Large, and Ex-Large.

TABLE II. 2003, DOY=301, H28V05: EXECUTION TIME (MINUTES)

	Stage-In	*Compute*	*Total*
Local	*0:44*	*5:46*	*6:32*
Medium (local)	*1:24*	*1:43*	*3:17*
Medium (blob)	*1:48*	*1:43*	*3:31*
Medium (UVa)	*13:35*	*1:43*	*15:38*
Large (local)	*0:59*	*1:46*	*2:47*
Large (blob)	*1:44*	*1:46*	*3:30*
Large (UVa)	*13:58*	*1:46*	*15:44*
Ex-Large (local)	*1:24*	*1:50*	*3:14*
Ex-Large (blob)	*1:53*	*1:50*	*3:43*
Ex-Large (UVa)	*13:47*	*1:50*	*15:37*

Our first task was to assess the performance of the Windows Azure instance sizes (note that from Table I it can be seen that 1 Ex-Large was approximately equal to 2 Large instances or 4 Medium instances). Table II shows the duration to analyze day-of-year (DOY) 301 on Windows Azure nodes of different size. For each size, we execute 3 configurations: "local" means that the input files are already on the virtual machine but not in the necessary folder; "blob" means that the input files are retrieved from Windows Azure Blob storage in the same datacenter; and "UVa" means that the input files are retrieved from the head node within UVa. The baseline is "local", which is the duration for one of the enterprise cluster nodes. The first column is the duration to copy the necessary input files into the local folder of the machine for the Matlab code. The second column shows the duration to execute the Matlab code. All durations are in minutes. This data shows:

1. Our science Matlab code executes at least 3 times faster than on our cluster nodes, even though the specifications are approximately equal. We believe that Windows Azure is under-promising its behavior, perhaps to ensure that it can meet its SLA under higher-duress situations.

2. As expected, all three VM sizes tested performed roughly the same for this single-threaded execution. In particular, the application is dominated by I/O and does not greatly benefit from the additional RAM of larger VMs.

3. Inside the cloud, input data should be retrieved from within the cloud. VPN functionality added convenience (to retrieve the input data from UVa), but wide-area latency/bandwidth makes any non-trivial I/O infeasible. Instead, pre-stage data as necessary to blob storage.

We next ran a more holistic experiment consisting of 16 DOY with a second stage reduction across different Windows Azure sizes (Table III). Our goal was to assess the duration of the escience experiment as well as its cost. Overall, the 16 DOY required 216 files (859 MB) be retrieved from Blob storage, and 118 files (1.58 MB) output were produced. Table III shows the duration and cost in dollars for a local compute node and three 8-core Windows Azure configurations. The cost is essentially just the VM cost, as it is free to upload data into Windows Azure and, once uploaded, it cost approximately $0.45 per month to store. We ignore the cost of Windows Azure boot/prep and assume that we delete the Windows Azure instances immediately after completion (in reality, these costs would be amortized across multiple runs).

We were initially surprised that the Medium VMs perform the best. Further analysis showed that with a 16-day computation and 8 cores, there were two 8-day "phases" that happened nearly in lockstep, as each DOY calculation took the same duration. At the beginning of each phase, 8 Matlab computations were each reading approx. 3 GB. With a single Ex-Large, all 8 computations were on the same machine. With the "Medium" configuration, only 2 processes were reading in parallel. Simply, the Medium VM and the Ex-Large VM had approximately the same I/O specs, so each Matlab computation took about *twice* the time on the Ex-Large VM (approx. 4 minutes compared to 2 minutes)! We found later that in larger experiments, eventually the concurrent DOY operations are not lockstep, so this effect is not as significant.

TABLE III. 2003, DOY=301-316, H28V05 (WITH STAGE 2 REDUCTION)

	Execution Time (minutes)	*Cost ($)*
Local (8 cores)	*21:50*	--
4 Medium	*10:14*	*$ 0.16*
2 Large	*12:07*	*$ 0.19*
1 Ex-Large	*14:17*	*$ 0.23*

TABLE IV. 2003, FULL YEAR,H28V05 (WITH STAGE 2 REDUCTION)

	Execution Time (1st Stage)	*Execution Time (2nd Stage)*	*Compute ($)*
Local (16 cores)	*2:33:16*	*4:24*	*$ 0*
32 Medium	*28:58*	*7:17*	*$ 3.70*
16 Large	*31:52*	*7:39*	*$ 4.07*
8 Ex-Large	*39:35*	*7:27*	*$ 5.06*
Hybrid: Local + 32 Medium	*28:12*	*7:20*	*$ 3.60*

We now evaluate (Table IV) whether to perform the full-year 2003 reduction using only our local enterprise cluster, cloudbursting using only Windows Azure nodes, or cloudbursting using both enterprise nodes and Windows Azure nodes. Rather than attempt to determine the cost per experiment today for our circa-2008 enterprise hardware, we approximate the marginal cost as $0. In all 5 cases, the Windows HPC queuing system is used to schedule the tasks. We decided to use 64 cores in Windows Azure – enough to substantially improve capacity beyond our existing infrastructure but not too many such that the experiment was unrealistically dominated by setup time. The number of cores to use for any particular escience experiment is the subject of future research. In terms of duration and cost, we see that 32 medium VMs alone is better than 16 large VMs alone or 8 ex-large VMs alone – our computation that takes 2 ½ hours in-house is completed in 35 minutes via cloudbursting with 32 medium VMs. It is not clear whether the hybrid case should be pursued, as the *overall* duration was greatly influenced by the relatively slow duration of the *last* DOY calculation on the local resources. In other words, the faster Windows Azure nodes were largely idle toward the end of the experiment as the last enterprise tasks were completing. Based on these experiments, we are currently using enterprise nodes to develop and test modifications to our satellite image processing algorithms. The small scale facilitates interactive debugging. We are also using the enterprise nodes when we need small-scale production runs (e.g., a few day-of-year calculations). When we need to run larger experiments, we cloudburst and acquire and use only Windows Azure nodes.

Based on our experiences, we recommend a number of steps when considering cloudbursting. First, benchmark the application in a broad number of situations on different cloud configurations. The cloud is largely a "black box", so it is very important to observe actual behavior rather than speculate. Second, focus on the application itself by minimizing "auxiliary" code whenever possible. For example, we greatly benefited by using the existing Windows HPC queuing system. Third, reduce (as much as possible) the heterogeneity between the local development environment and cloud platform. Finally, we believe that a stage-in, execute, stage-out pattern fits well with cloudbursting and that it is not as clear for other I/O patterns. We have found that in general that the determining factor is data – where it is and how much is moved. In many situations, the key to successful cloudbursting is to minimize data movement.

VI. CONCLUSION

For the near future, a significant challenge for enterprises is how best to explore the potential of cloud computing while continuing to leverage their existing computational infrastructure. In this paper, we have investigated a concrete example of cloudbursting as a means to produce new scientific results through satellite image processing. We found that the bag-of-tasks model of our escience application greatly benefited from cloudbursting, as we achieved good performance, software development productivity was enhanced, and a cloud-based execution was cost-effective. We believe that further specific studies like this one are needed, as inevitably different factors will most likely determine the appropriateness of cloudbursting for different applications.

REFERENCES

[1] 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 Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010), Apr 19-23, 2010. Atlanta, Georgia.

[2] 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. Dec 7-10, 2010.

[3] C. Lee, "A Perspective on Scientific Cloud Computing", 1st Workshop on Scientific Cloud Computing (ScienceCloud 2010), Chicago, IL, June 21 2010.

[4] The Florida Stratus Cloud: http://www.acis.ufl.edu/vws/.

[5] P. Watson, P. Lord, F. Gibson, P. Periorellis, and G. Pitsilis, "Cloud Computing for e-Science with CARMEN", the 2nd Iberian Grid Infrastructure Conference, Porto, May, 2008.

[6] A. Matsunaga, M. Tsugawa, J. Fortes. CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications Proceedings of 4th IEEE International Conference on e-Science (eScience 2008), Indianapolis, IN, Dec 2008.

[7] C. Evangelinos and C. Hill. Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. Cloud Computing and its Applications (CCA-08). Chicago, IL. Oct 22-23, 2008.

[8] R. Grossman, Y. Gu, Data mining using high performance data clouds: experimental studies using sector and sphere. Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'08). Las Vegas, NV, 2008.

[9] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, J. Wang. Introducing Map-Reduce to High End Computing. 3rd petascale data storage workshop (held in conjunction with SC'08). Austin, TX. Mon Nov 17, 2008.

[10] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good. The Cost of Doing Science on the Cloud: The Montage Example. Proceedings of Supercomputing 2008, Austin, TX, Nov 15-21, 2008.

[11] K. Jackson, L. Ramakrishnan, K. Runge, and R. Thomas. "Seeking Supernovae in the Clouds: A Performance Study". 1st Workshop on Scientific Cloud Computing (ScienceCloud 2010), Chicago, IL, June 21 2010.

[12] M. Palankar, A. Lamnitchi, M. Ripeanu, S. Garfinkel. Amazon S3 for Science Grids: A Viable Solution? International Workshop on Data-Aware Distributed Computing, Boston, Massachusetts, June 2008.

[13] W. Lu, J. Jackson, and R. Barga. "AzureBlast: A Case Study of Developing Science Applications on the Cloud." 1st Workshop on Scientific Cloud Computing (ScienceCloud 2010), Chicago, IL, June 21 2010.

[14] W. Lu, J. Jackson, J. Ekanayake, R. S. Barga, N. Araujo. Performing Large Science Experiments on Azure: Pitfalls and Solutions. IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010), Nov. 30-Dec.1, 2010, Bloomington, Ind.

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

[16] Z. Hill and M. Humphrey. 2009. A Quantitative Analysis of High Performance Computing with Amazon's EC2 Infrastructure. In Proceedings of the 10th IEEE/ACM International Conference on Grid Computing (Grid 2009). (Oct 13-15 2009).

[17] E. Walker, "Benchmarking Amazon EC2 for High-Performance Scientific Computing", ;Login: The Usenix Magazine, Vol. 33, No. 5., 2008

[18] 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. IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010), Nov. 30-Dec.1, 2010, Bloomington, Ind.

[19] Candeia, D.; Araujo, R.; Lopes, R.; Brasileiro, F..Investigating Business-Driven Cloudburst Schedulers for E-Science Bag-of-Tasks Application. IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010), Nov. 30-Dec.1, 2010, Bloomington, Ind.

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

[21] 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): 1313-1323, 1998.

[22] Huete, A., Didan, K., Miura, T., Rodriguez, E. P., Gao, X., & Ferreira, L. G. (2002). Overview of the radiometric and biophysical performance of the MODIS vegetation indices. Remote Sensing of Environment, 83, 195– 213

[23] W. E. Esaias, M. R. Abbott, I. Barton, O. B. Brown, J.W. Campbell, K. L. Carder,D.K. Clark, R. H. Evans, F. E. Hoge, H. R. Gordon,W. M. Balch, R. Letelier, and P. J. Minnett, "An overview of MODIS capabilities for ocean science observations," IEEE Trans. Geosci. Remote Sensing, vol. 36, pp. 1250–1265, July 1998.

[24] Z. Hill and M. Humphrey. CSAL: A Cloud Storage Abstraction Layer to Enable Portable Cloud Applications (work in progress). 2nd IEEE International Conference on Cloud Computing Technology and Science . November 30-December 3, 2010 . Indianapolis, Indiana.