eScience in the Cloud: A MODIS Satellite Data Reprojection and Reduction Pipeline in the Windows Azure Platform

Jie Li*, Deb Agarwal**, Marty Humphrey*, Catharine van Ingen***, Keith Jackson**, and Youngryel Ryu****
* Department of Computer Science, University of Virginia, Charlottesville, VA USA
** Lawrence Berkeley National Lab, Berkeley, CA USA
*** Microsoft External Research, San Francisco, CA USA
****Dept. Environmental Science, Policy and Management, UC-Berkeley, Berkeley, CA USA

Abstract— **The combination of low-cost sensors, low-cost commodity computing, and the Internet is enabling a new era of data-intensive science. The dramatic increase in this data availability has created a new challenge for scientists: *how to process the data*. Scientists today are envisioning scientific computations on large scale data but are having difficulty designing software architectures to accommodate the large volume of the often heterogeneous and inconsistent data. In this paper, we introduce a particular instance of this challenge, and present our design and implementation of a MODIS satellite data reprojection and reduction pipeline in the Windows Azure cloud computing platform. This cloud-based pipeline is designed with a goal of hiding data complexities and subsequent data processing and transformation from end users. This pipeline is highly flexible and extensible to accommodate different science data processing tasks, and can be dynamically scaled to fulfill scientists' various computational requirements in a cost-efficient way. Experiments show that by running a practical large-scale science data processing job in the pipeline using 150 moderately-sized Azure virtual machine instances, we were able to produce analytical results in nearly 90X less time than was possible with a high-end desktop machine. To our knowledge, this is one of the first eScience applications to use the Windows Azure platform.**

Keywords-cloud computing; eScience; large-scale data-intensive applications

1. Introduction

The combination of low-cost sensors, low-cost commodity computing, and the Internet is enabling a new era of data-intensive science. Many scientists are envisioning analysis and synthesis computations that can easily go beyond tera-scale or even peta-scale data capabilities. For example, remote sensing is one of the factors transforming environmental science today. Emerging inexpensive ground-based sensors are enabling scientists to make field measurements of science variables at more locations with more spatial and temporal granularities. Satellites and other remote sensing technologies can at times provide a second source of measurements to the data generated by the ground sensors, and at other times serve as the single source of data for those parts of the Earth that are not amenable to ground-based sensors. Large-scale environmental data is necessary to investigate global-scale phenomena such as global warming.

Given these remarkable technological improvements, scientists can increasingly find the sensor data they need on the Internet, or they can design and deploy a custom software architecture to generate the sensor data they need. But now scientists are facing with a new challenge: *how to process the data*. First, the source data from different locations or measurements can be very heterogeneous in data format, resolution, time frame, confidence, etc., making it difficult for scientists to use directly in their research. Second, deriving from this first issue, the best algorithms can often not be determined until the raw data is actually available, because the algorithms designed under ideal situations often do not account for the less-than-ideal raw data. Third, the inherent nature of the scientific hypotheses and experiments often require a *scale* beyond which the scientists have previously encountered. A scientist's commodity workstation will quickly be overwhelmed for non-trivial computations over large datasets. Purchasing larger computational platforms (i.e., clusters) – although arguably the norm – is generally not cost-effective even if the money is available, as they quickly become outdated and especially if additional system administrators must be hired by the domain scientists just to manage the infrastructure. Obtaining an account and allocation on national-scale supercomputing centers can be pursued but frequently offers a particular programming and debugging environment that is not as flexible as a local option, and user jobs can sometimes require a potentially long wait in a queuing system.

This paper is motivated by a particular instance of this problem, as we attempted to integrate data from ground-based sensors with the Moderate Resolution Imaging Spectroradiometer (MODIS) [1][2] satellite data. Designed to improve the understanding of global dynamics and processes occurring on the land, oceans, and lower atmosphere, the MODIS data is generated by the *Terra* and *Aqua* satellites and is a viewing 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 are using the MODIS data to explore and validate scientific hypotheses (e.g., see [3] for an overview with regard to vegetation and [4] for an overview with regard to ocean science). We encountered all three issues identified above, particularly the inability to practically compute such data transformation and integration on a scientist's desktop machine. As for our example, even a high-end desktop machine with Quad-core processing would require *tens of months* of continuous processing to compute our result (details are provided later in this paper). Furthermore, our desire to expose this as a Web service for the broader environmental research community could not possibly be implemented if we used a desktop or cluster based approach.

In this paper, we describe our experiences designing and implementing our MODIS data integration and transformation techniques by using cloud computing, specifically the Windows Azure platform [5]. To our knowledge, this is one of the first science applications to use the Windows Azure platform. The contributions of this paper are:

- We present a novel approach to "reproject" the input data into timeframe- and resolution-aligned, and uniform geographically formatted data.
- We present a novel "reduction" technique to derive important new environmental data through the integration of satellite and ground-based data.
- We describe how we leveraged the Windows Azure abstractions and APIs to accomplish the reprojection and reduction steps.
- We present our general observations and lessons learned as we explore how emerging common cyberinfrastructure such as Windows Azure can be leveraged by resource-constrained domain scientists.

While cloud computing is still in its infancy and many challenging open issues remain, we believe that the Windows Azure platform is a compelling approach for large-scale scientific explorations, as, for example, we gained a factor of 90x speedup over a high-end desktop machine when running 150 Azure virtual machine instances for parallel data reprojection tasks. Although the project was developed specifically for MODIS data processing, we believe that the experience we gained and lessons learned from implementing this architecture in Windows Azure can be applied to a broad range of other imaging eScience applications.

The remainder of this paper is structured as follows. Section 2 discusses existing use of cloud computing for eScience. Section 3 gives an overview of Windows Azure. In section 4 we introduce the background of our project. Section 5 introduces the implementation details of our pipeline architecture. Section 6 shows the evaluation results of the pipeline. In section 7 we conclude our work with discussions on our experience with Windows Azure and some general observations and unresolved issues about leveraging cloud computing for eScience.

2. Existing Use of Cloud Computing for eScience

Commercial cloud computing platforms such as Amazon's Elastic Compute Cloud (EC2 [6]), Google App Engine [7], and Microsoft's Windows Azure have been created to offer highly flexible, scalable, and on-demand computational, storage, and networking resources for large-scale computing tasks. It can be argued that the emphasis of these commercial clouds is at least perceived to be for "business applications", so it is not clear how readily such commercial clouds can be used for *science*. We believe such platforms have the potential to be a cost-efficient computing platform for domain scientists, particularly due to the economies of scale and increasing market competition, which will drive the costs down. Furthermore, the pay-as-you-go manner in the business model of cloud computing means scientists no longer need any specific authorization for using powerful computing resources and there is no large up-front cost.

The number of Cloud-based science applications has been increasing. Evangelinos and Hill pursue the use of EC2 for Atmospheric-Ocean models [8]. The CARMEN e-science project is designing a system to allow neuroscientists to share, integrate and analyze data, and has been recently expanded to include cloud computing [9]. Keahey *et. al.* describe early experiences to deliver EC2-like cycles to scientific applications [10]. The feasibility of executing workflows in the cloud is being studied, particularly for the Montage application [11]. (Montage is the subject of another cloud-based analysis in [12]). Analysis of gene expression data and microarrays is being pursued in public clouds [13]. Other studies are beginning to appear that address the performance of science applications in the cloud [14][15]. While these efforts and others are pioneering the use of science applications in the cloud, we believe that there are many unresolved issues on how, in particular, domain scientists can best leverage and exploit cloud computing. We believe our cloud-based environmental data analysis reported in this paper complements this existing research by reporting the first experiences using Windows Azure for eScience.

3. Windows Azure overview

Windows Azure was announced by Microsoft as its cloud computing service platform in Microsoft Professional Developers Conference (PDC) 2008. Early access in the Community Technical Preview phase started from spring 2009. Commercial availability of Windows Azure has started since January 2010.

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 EC2, Windows Azure does not expose the virtual machine abstraction directly to an end-user. In EC2, users can customize the environment for their particular application by installing particular software or by purchasing particular machine images. Windows Azure achieves flexibility via the wide range of language supports other than the .NET framework, such as Java and PHP. Windows Azure supports popular Internet standards and protocols including SOAP, REST, XML, etc.

In Windows Azure, the virtual machine instances can be classified as two different roles: the web service hosting instances called *Web Roles* and the computational instances called *Worker Roles*. Developers can specify the number of instances for both roles at the deployment of their application, or can dynamically adjust the number of instances at runtime.

Besides the computational resources provision, Windows Azure also provides three types of cloud storage services:

- *Blob service*, which provides durable and scalable storage service for 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.

4. MODIS Satellite Data Processing

As mentioned earlier, the MODIS data is generated by the *Terra* and *Aqua* satellites and is a viewing of the entire Earth's surface in 36 spectral bands, at multiple spatial resolutions, generated every 1-2 days. MODIS provides various biophysical variables (e.g. gross carbon uptake, albedo etc) with spectral irradiance ranging visible, near infrared, infrared and thermal regions of the electromagnetic spectrum. It has been an important scientific source and can be applied to many environmental studies from local to global scale. The MODIS data is made available every day over the Internet on multiple FTP sites.

There are three main barriers for scientists to obtain the L2/L3 MODIS data and apply analyses on them to produce scientific results in their research:

Data Collection: The L2/L3 data is separated into 3 main groups: Atmosphere, Land, and Ocean. Each data group includes a number of different data products, which are published and maintained on different FTP sites. For each day the L2/L3 data are stored in tens to hundreds of separate pieces of files and the corresponding metadata such as earth covering boundaries, observing time, etc. are kept in a different place. Although there is a web portal [16] for data download by queries (geographic area, time period, data products, etc.), it only supports queries for a very limited data size. If a scientist wants to download a larger size of the L2/L3 data, he/she needs to download them directly from the FTP sites, which is almost infeasible given the data size is large and no tools are available for metadata query in order to decide which set of data files to download.

Data Transformation: Data products from different groups are stored in different geographical project types. For example, atmosphere data products are directly derived from the MODIS instrument and use a *swath* space and time system that follows the satellite, while the land data products have already been preprocessed and use a gridded space and time coordinates called the *sinusoidal* tiling system. The reprojection from one type to the other involves very complex data processing algorithms, and requires in-depth programming skills which are not common to the domain scientists. A number of tools [17][18] have been developed by the community to perform the reprojections. However, these tools only work on a single data file as input and lack support for processing/producing a customized area and time period of data. Thus, these tools are of limited value when scientists need to synthesis a large spatial and time scope of data. Furthermore, data from different products can have different time frames and space resolutions, which also need to be unified before scientists could conduct scientific computation on them. This data heterogeneity makes it extremely difficult for scientists to use data from different products directly in their analytical processes, and a main data transformation step is required to produce timeframe- and resolution-aligned, and uniform geographically projected data.

Data Management: To date, most scientific research involving the MODIS data are limited to a small geographical area. It is not feasible to conduct scientific research in continental or global scope not only because of the above two barriers, but also because the data management cost would become overwhelming. For example, the research reported in this paper is motivated in part by a group of biometeorologists needing to conduct a MODIS data synthesis involving nine data products covering the whole US continent in a 10 year time frame. Table I contains a description of the nine data products, including the total size of the L2/L3 data the scientists will have to manage. *#Source Data Files* indicates the total number of source data files that need to be downloaded for each data product for the whole US over 10 years, and *Data Size* indicates the size of each data product that needs to be downloaded from the multiple FTP sites. Prior to the research reported in this paper using Windows Azure, these scientists would first need to query the metadata to decide the subset of source data files which covers the US area, download all the source data files to their local storage, reproject the 5 Swath type products into Sinusoidal type, unify the timeframe and space resolution of all 9 data products, and finally perform scientific computation on the data. This process would need to be managed and performed manually by these scientists. This process as a whole is very time-consuming, tedious and error-prone.

TABLE I. DATA PRODUCT REQUIREMENTS FOR US CONTINENT OVER 10 YEARS

	MO(Y)D04L2	**MO(Y)D05L2**	**MO(Y)D06L2**	**MO(Y)D07L2**	**MCD12Q1.005**
Data Group	Atmosphere	Atmosphere	Atmosphere	Atmosphere	Land
Project Type	Swath	Swath	Swath	Swath	Sinusoidal
#Source Data Files	65K	130K	130K	130K	150
Data Size	85GB	500GB	2TB	850GB	13GB
	MCD15A2.005	**MCD43B3.005**	**MO(Y)D11_L2.005**	**MO(Y)D13A2.005**	
Data Group	Land	Land	Land	Land	
Project Type	Sinusoidal	Sinusoidal	Sinusoidal	Swath	**Total**
#Source Data Files	7K	7K	113K	3K	585K
Data Size	13GB	110GB	337GB	40GB	3.9TB

Even if the above process were manageable and undertaken by scientists, there is still a significant challenge in finding/obtaining the necessary computational capacity to perform the analysis. An entire end-to-end computational run for the US continent over 10 years would require *tens of thousands* of CPU hours of continuous processing. Clearly, the MODIS data presents unprecedented opportunity for scientific exploration, but there remain significant practical issues that prevent such discoveries, particularly for resource-constrained scientists.

Therefore, to help these and other scientists, we have designed and implemented a MODIS data reprojection and reduction pipeline in the Windows Azure platform. We choose to leverage cloud computing to provide on-demand computational resources and highly scalable infrastructure for our data processing pipeline based on the observation that the MODIS data files can be processed largely in parallel. Thus we can map the reprojection or scientific computation tasks on these data files to a large number of parallel computational instances in the cloud. Through the implementation of this pipeline, our goals have been to provide the following main contributions for scientists pursuing reprojection/reduction of the MODIS data:

- Reduce or eliminate the issues involved in data collection, transformation, and management processes by constructing an automatic data processing and management framework in Windows Azure. The scientist can make requests to the pipeline through a web portal by specifying their data requirements, which will then be automatically processed by the pipeline to collect, reproject, integrate the source data, and then perform analytical computation on them. The scientist will receive email notifications on the start and completion of the requested data processing job. The completion notification email also includes a single download link to the data result package. During the execution, scientists can monitor the status detail of each single task and view execution logs at real-time through the web portal.
- Leverage cloud computing to provide a highly-flexible and scalable data processing pipeline architecture. The number of computational instances in the cloud can be adjusted at runtime to support different scales

of data processing tasks, thus to accommodate continental or even global scope of MODIS scientific research.

- Scientists can specify a custom *reduction* step, in which scientists will perform synthesis and analyses computation on the unified data produced after the reprojection step, and will usually reduce these data into a much smaller set of final science data results. We enable our scientists to specify different reduction algorithms in their own tools such as compiled MATLAB executables or other command-line tools. They can be uploaded through the request submission web page and then executed by the computing instances in Windows Azure.

We believe that this reprojection and reduction service of the MODIS data is desired by a number of scientists today who would otherwise be unable to utilize the MODIS data. Furthermore, we believe our architecture can also be applied for the processing of data other than the MODIS remote sensing data.

5. Pipeline Implementation

5.1 Pipeline Stage Overview

Our MODIS data reprojection and reduction pipeline consists of three main stages as shown in Figure 1. Each stage is an independent component and addresses a specific type of data-centric tasks:

Data collection stage: At this stage the data collection model assembles all the required source data according to a specific user request. Each task first determines and then downloads all of the necessary source files for the specified target images. Azure tables are used to hold the metadata describing the FTP site, the remote file structure, and geographical information about the source files. Once the metadata for the specified source files are queried from the meta-tables, the working Azure instance first checks whether the source files have already been downloaded and exist in the Data Repository in Azure blob storage (as explained and discussed later). If they are already in the data repository, the Azure computing instance will directly copy them from the blob storage to its local disk storage for the reprojection computation. If they don't exist in the data repository, then the instance will have to download the files from the FTP sites to local storage as for the reprojection, and then also upload them to the data repository to be reused for future computations. When all the source data is available on local storage, the instance will then initiate the reprojection computation for that target.

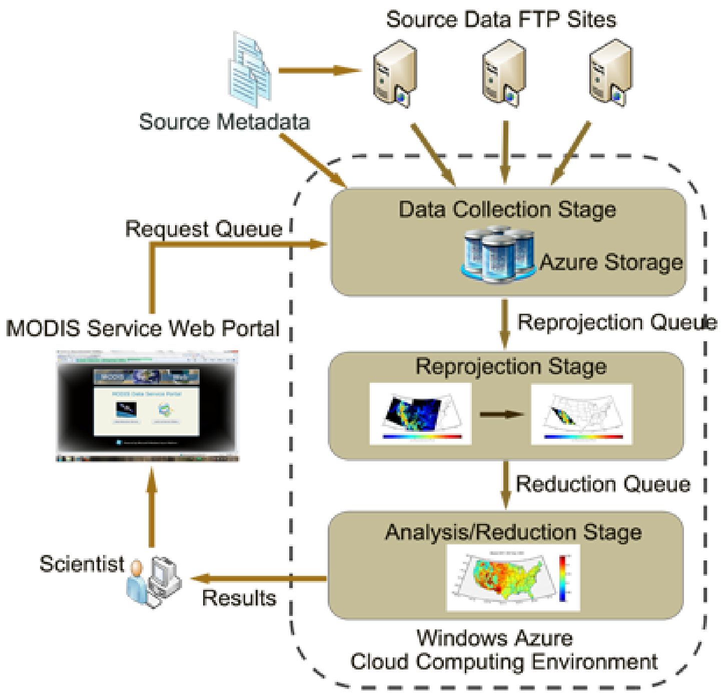

Figure 1. Pipeline stage overview

Reprojection stage: This stage performs reprojection tasks on the source data collected in the first stage. A single task produces exactly one reprojected target image. While the total number and size of data processing requirements for the whole computation request can be substantial (e.g. compute reprojected data on US continent for a whole year), the computation requirement for a single task is relatively small, as the whole request can be separated into a large number of embarrassingly parallel small tasks. Typically, a single reprojection task requires a collection of 3-4 source data files, each in the size of several MBs to tens of MBs. The reprojection computation can be finished in several minutes on a single Azure computing instance.

Like the source data, the reprojected data can also be reused across multiple requests. Once a single reprojected image has been computed and uploaded to the data repository, it will be cached for future reuse. Therefore, when a computing instance receives a new reprojection task, it will first check whether the result data has already been computed before. If the target data already exists in the data repository, it simply aborts the task and uses the result in the data repository.

Analysis and reduction stage: The (optional) reduction stage works on the data produced by the reprojection stage to perform custom analysis algorithms as specified by the scientists. Each task may take any number of input source files (specified as different aggregation types by scientists), and produces a number of reduced image files. This flexibility and extensibility of choosing customized analysis tools has been a major advantage of our architecture to provide support for a broad range of scientific applications, rather than a hard-coded one. It is very convenient for domain scientists to update or change the algorithms for the analysis across multiple requests, which can be especially efficient and helpful in scientific code development phase. Based on our experience, it has saved us a lot of time on system maintenance as scientists may go through a large number of iterations during the development of scientific algorithms without the assistance of a system administrator.

Aside from this flexibility, our current implementation places some limits on the reduction algorithm that is uploaded by the scientist. First, the reduction tool must be a command line executable and all custom libraries other than the .NET framework must be included with the tool. Second, since data transfers between the local disk storage of computing instances and the data repository in Azure blob storage are automatically managed by our software framework, the scientist will need to specify the local paths in his/her code for the input and output data images or just use the given default local input and output paths. After the reduction algorithm is executed, the output data files will be uploaded to the data repository in Azure blob storage, and a notification email will be sent to the requester which includes the download link for the results.

5.2 Data Repository

The data repository in Azure blob storage serves both as the long-term and short-term cache for all sharable data, including the source data, intermediate reprojected data, and final result data. All these data are stored as blobs and grouped in different blob containers. Users don't need to worry about the details such as semantic groupings and locations of their data. Using their request IDs, they can simply retrieve and manage their data through the web portal.

As discussed above, both the source data downloaded from FTP sites and the intermediate reprojection results are cached in the data repository to enable reuse across multiple requests. This mechanism can save network bandwidth by avoiding downloading the same source data for multiple times, and can save large amounts of CPU hours which would otherwise be used to re-compute the same reprojection results. However, we need to be aware that current commercial cloud computing platforms not only impose charges on network bandwidth and computing resource usage, but also charges on persistent storage usage. Therefore, whether it is a right decision to use data caching will depend on the nature of the scientific applications. Especially, tradeoffs need to be made between the cost of storage usage of data cache and the cost of re-downloading the source data and re-computing the results every time. For the application we are currently working on, it would be beneficial to maximize reuse of data, since it would take a large amount of CPU hours to reproduce the data results every time. A detailed discussion and case study on this issue can be found in [12].

An important design aspect that has not been incorporated into the implementation to date is security with regard to intermediate data results, final data results, and uploaded reduction algorithms. We plan to implement fine-grained access control for this important data in a future version of the pipeline by leveraging the .NET Access Control Service [19] in the Windows Azure platform.

5.3 Task Scheduling

We based our task scheduling and execution model on the Azure queue services, which is a loosely coupled message dispatching and communication service. Each pipeline stage has a specific number of Azure computing instances which keep fetching and executing tasks from the corresponding task queue. A service manager is running in the pipeline to accept user requests from the request queue. It then parses and separates the request into a number of parallel tasks, and dispatches them to the different task queues. The service manager also monitors and updates the status of each single task during the computation, and will take necessary correction actions when a task runs into a faulty state (discussed in the next part). We also provided a job monitoring web service on the web portal so that scientists can look up the status of their requests in real-time.

Figure 2 shows the implementation detail of the reprojection stage only. A typical reprojection computation includes the following steps:

1. The scientist sends a reprojection request through the web portal to the service request queue.

2. The service manager fetches the request from the request queue, parses it into a number of parallel data collection and reprojection tasks. It then dispatches the tasks to the data collection and reprojection task queues. At the same time, it also registers the information of this new request and status of all single tasks in a service log table. It then starts monitoring and updating the status of the reprojection request.

3. Each of the computing instances in the reprojection stage keeps fetching and executing the reprojection tasks from the task queue. When the worker gets a task from the queue, it first parses the information for that single reprojection task, queries a number of meta-tables for the corresponding source data items, and then downloads those data files from the blob storage into temporary local storage. It then reprojects the source data files into intermediate data results in the format as specified by the user request. When the computation is done, it uploads the result to the data repository, and then updates the information for that task in the service log table.

4. When all tasks for a specific request are finished, the service manager will send the requester a notification email which includes the download link to the result data package.

5. When the scientist receives the notification email, he/she can either download the reprojected data to a local machine and perform further analytical computation locally, or specify another reduction request to do analytical computation on that set of data in Windows Azure. However, in that case, it would be more convenient for the scientist to combine the reprojection and reduction requests into a single request.

The task scheduling for the data collection and reduction stages are similar to the reprojection stage, thus we will not introduce their implementation details here.

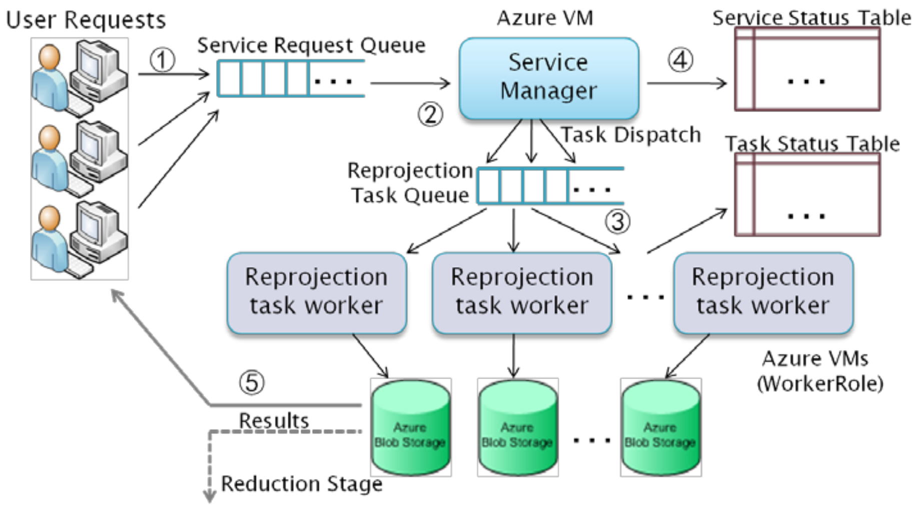

Figure 2. Task scheduling in the reprojection stage

5.4 Fault Tolerance

Fault tolerance is an important and even a critical part of many large-scale distributed systems. In our architecture, the service manager keeps tracking the status of each single task and takes fault-tolerant actions when necessary. If a computing instance crashes during execution or the task running on that instance doesn't finish after a default time limit, the same task will be sent to the task queue again and restarted by another instance. However, if that task still fails after totally 3 retries, it is most likely that the failure is caused by some intrinsic errors such as reprojection on crashed source data files. Thus the service manager will give up the retry and indicate the task as a failure. Usually, scientists will request for a computation job which consists of a large amount of tasks, and a single failed task doesn't necessarily mean the whole final results will become invalid.

6. Evaluation

We evaluated the performance of running the pipeline stages in Windows Azure using an account which has a quota of 200 computing instances. Of all 3 stages, the reprojection stage is most compute- and data-intensive, and also involves the most frequent usage of all 3 types of Azure storage services (blob, queue, and table service), thus we choose the reprojection task processing for our performance and scalability test.

6.1 Single instance performance

We first compare the performance of a single computational instance in Windows Azure with that of a local high-end desktop machine. Table II shows the capacity of the desktop machine we used for the test and the equivalent capacity of a single small-size Azure computing instance. We did our experiment by executing totally 45 different reprojection tasks on both the desktop machine and a single Azure instance. Each reprojection task involves the following operations: 1 Azure table query to get the list of source data files that are required for the given reprojection task; 4~8 table queries for the metadata of each source data file; downloading all source data files from Azure blob storage to local storage; performing reprojection computation on downloaded source data and generating the reprojected data.

TABLE II. DESKTOP MACHINE AND AZURE INSTANCE CAPACITIES

	Desktop	**Azure Instance**
Capacity	CPU: Intel Core2Duo E6850 @ 3.0GHZ Memory: 4GB Hard Disk: 1TB SATA Network: 1Gbps Ethernet OS: Windows 7 RC Build7100 (32-bit)	CPU: 1.5-1.7GHZ X64 equivalent processor Memory: 2GB Local Storage: 250GB Network: 100Mbps OS: Windows 2008 Server x64 (64-bit)

TABLE III. REPROJECTION EXECUTION STATISTICS

a) MOD04L2

	Avg. T_W	**Avg. T_C**	**Avg. T_R**
Desktop	39.1s	9.2s	29.9s
Azure Instance	82.7s	14.9s	67.7s
Perf. Ratio	2.1	1.6	2.3
SD(Perf. Ratio)	0.21	0.79	0.13

b) MOD06L2

	Avg. T_W	**Avg. T_C**	**Avg. T_R**
Desktop	169.5s	65.8s	103.7s
Azure Instance	267.5s	52.6s	214.9s
Perf. Ratio	1.6	0.80	2.07
SD(Perf. Ratio)	0.13	0.31	0.07

c) MOD11L2.005

	Avg. T_W	**Avg. T_C**	**Avg. T_R**
Desktop	80.3s	14.9s	65.4s
Azure Instance	151.5s	15.3s	136.1s
Perf. Ratio	1.9	1.0	2.1
SD(Perf. Ratio)	0.39	0.73	0.40

In our experience, the time required for the reprojection tasks of different products can differ widely due to different characteristics of the source data. Thus we selected the 45 reprojection tasks from 3 different MODIS data products (15 tasks for each): MOD04L2, MOD06L2, and MYD11L2.005, to account for the variability across different data product characteristics. For each task, we measured its *wall clock time (T_W)*, which is the period from when the Azure computing instance or the desktop machine starts processing the task to the time when it finishes the reprojection computation. We then broke down the total wall-clock time into two parts:

- *Communication time (T_C).* It is the time used for querying metadata in Azure tables and downloading source data files from the data repository in Azure blob storage to local disk storage.
- R*eprojection time (T_R).* It is the time used to perform reprojection computation on the downloaded source data files.

Table III(a)-(c) show the average results of the execution of all 45 tasks on both the desktop machine and the Azure instance. Each table shows statistics for the 15 reprojection tasks from each data product. *Avg. T_W* shows the average wall clock time (in seconds) for a single task across all 15 tasks. Similarly, *Avg. T_C* and *Avg. T_R* show the average communication time and reprojection computation time across all 15 tasks. The first two rows in each table show the comparison of the results for the desktop machine and the Azure instance. The *Perf. Ratio* shows the performance ratio of the two as calculated by *Azure Instance Time* / *Desktop Time*. Finally, *SD(Perf. Ratio)* shows the standard deviation of the performance ratio across all 15 tasks.

It is not surprising to see that, in all cases, the task execution time as indicated by T_W of the desktop machine is always less than that of the Azure instance by a factor of 1.6~2.1. This is because the raw hardware capacity of the desktop machine is much better than the Azure instance.

The communication time results illustrate some interesting behaviors. First, the Azure storage communication performance ratios of Azure instance and desktop machine are not consistent across all 3 data products. Second, the standard deviation of communication performance ratio is very large and can account for as much as 70% of the performance ratio itself. To further demonstrate this behavior in detail, Table IV shows a detailed list of communication time measurements from all 15 tasks in MYD11L2.005 product, which has the largest standard deviation as of the percentile of the performance ratio itself. As we can see, the communication performance ratios for each of the 15 tasks present a large factor of randomness. This indicates that the communication time is mostly affected by the highly inconsistent response time and data transfer rate on the Azure Storage Servers, rather than the Internet connection bandwidth. Furthermore, although the Azure instances are running in the same datacenter with the storage for our source data (which can be specified using the so-called *Affinity Group*), it seems these instances didn't share any benefits from its internal network connection advantages because its performance is not even as good as a machine connected from outside the datacenter.

On the other hand, the statistics on the reprojection computation time shows a much more consistent pattern, which indicates that the virtualized hardware resource capacity for each Azure instance is much more stable in application execution performance.

TABLE IV. COMMUNICATION TIME FOR ALL 15 TASKS IN MYD11L2.005 (UNIT: SECONDS)

Task	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Desktop	12	16	14	18	19	9	21	16	7	27	14	6	23	6	16
Azure Instance	7	18	17	18	17	16	16	19	16	20	19	5	18	19	5
Perf. Ratio	0.58	1.13	1.21	1.00	0.89	1.78	0.76	1.19	2.29	0.74	1.36	0.83	0.78	3.17	0.31

6.2 Parallel Execution Performance and Scalability

One of the main advantages of cloud computing is the high flexibility and scalability to provide on-demand computing resources. Our pipeline architecture is very flexible in terms of the ability to dynamically adapt to the computational requirements of different data processing requests. To see how the performance of our pipeline scales with the number of the computing instances, we measured the total processing time for running 1500 reprojection tasks from each of the above 3 data products using different number of Azure instances. For each data product, we run the same 1500 tasks using 50, 100, and 150 Azure instances, and measure the total processing time for all tasks. We then estimated the speedups we got as compared to the capacity of a single Azure instance and that of our high-end desktop machine. Table V shows the total processing time (in hours) for each run. Azure pipeline runs with 50, 100, and 150 instances are all measured from real tests, while the single Azure instance and desktop processing time is only an estimation, since it would take days to finish the whole run. To calculate the single Azure instance and desktop processing time for the 1500 tasks for each data product, we first get the average wall clock time for a single task as measured in Section 6.1, and then multiply it by 1500.

From the table we can calculate the speedups for running multiple Azure instances over a single instance and a single desktop machine. The results are shown in Figure 3. The "VMs" in the figure indicates the Azure computing instances. As the figure shows, the performance of our pipeline scales almost lineally with the number of instances, and although each Azure instance has a moderate capacity as compared to the high-end desktop machine, in best case we could still get a nearly 90x speedup over a single desktop when running tasks for MOD06L2 with 150 Azure instances. As compared to the performance of a single Azure instance, we almost get the same times of speedups as the number of instances in all cases except for product MOD04_L2.

TABLE V. PROCESSING TIME FOR 1500 TASKS (UNIT: HOURS)

	MOD04_L2	MOD06_L2	MYD11_L2.005
Desktop	16.29	72.62	33.45
Single instance	34.21	116.19	63.56
50 instances	0.76	2.25	1.12
100 instances	0.40	1.20	0.61
150 instances	0.30	0.85	0.44

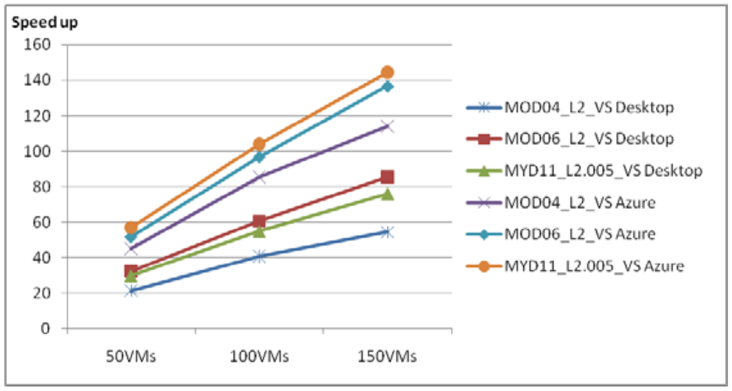

Figure 3. Performance speedups using parallel instances

6.3 Scalability of the Azure Storage Service

Since there are many Azure instances running concurrently in the pipeline, they will frequently perform concurrent operations such as table queries, queue/dequeue operations, blob uploads/downloads, etc. thus we expect that the number of concurrent working instances in Azure will impact the performance of Azure Storage Services. To measure how well the Azure Storage Services scale with the number of concurrent instance requests, for each of the reprojection runs on product MOD06L2 in Section 6.2, we also accumulated the total Azure storage service communication time and the computation time for all 1500 tasks. We choose product MOD06L2 for our measurements because reprojection tasks for this product have largest download size requirement (~100MB for each task), and thus can present the severest concurrent operation request behaviors for Azure Storage Services.

Figure 4 shows the accumulated reprojection time and communication time for all 1500 tasks for MOD06L2 product. With 50 Azure instances running concurrently, the total Azure Storage Service communication time is a little less than 20 hours, while it becomes more than 27 hours when 150 instances are running concurrently. This result indicates that the scalability is not so free when it comes to the Azure cloud storage service, and the latency can get significantly larger when there are many concurrent data requests to the same queue, table, and blob item in the storage.

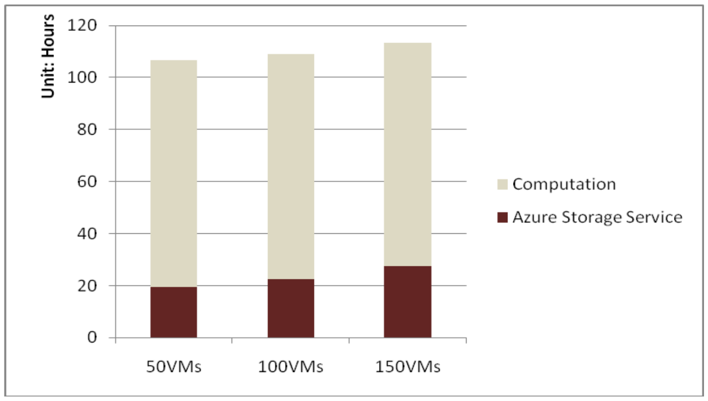

Figure 4. Total Reprojection time VS. Communication time

7. Conclusion

In this paper, we have described how we leveraged cloud computing to create a MODIS satellite data reprojection and reduction pipeline. We are currently undergoing user studies involving biometeorologists, in part to validate the results from our first processing of a single year covering the whole US continent. We expect to launch the remaining 10 year computation for the US shortly and look forward to global scale. With the flexibility and scalability provided by the cloud computing model, scientists who previously only worked on their desktop machines can now access powerful computing resources and to scale their research scopes in a cost-effective way. Our implementation of the MODIS data processing pipeline in Windows Azure has further demonstrated the potential of using cloud computing to lower a series of data entry barriers for eScience by leveraging the scalable storage and computing services.

Through the design and implementation of this scientific application, we have first-hand experience with a number of issues for the broader eScience community. Commercial cloud computing infrastructures today are, to a large extent, running behind the scene. This infrastructure transparence makes it more difficult for cloud application developers to diagnose problems as they occur. In our experience, troubleshooting problems in cloud applications is not only time-consuming, but sometimes can also be very tricky. As a result, managing code dependencies took a noticeable part of our development time. Local simulation and debugging for cloud applications is another remaining challenge. While Windows Azure provides a local simulation environment for cloud application development, there are many cases that applications will run into different behaviors when the application is deployed, which is due to the heterogeneity of the underlying software and hardware environments where the cloud applications are actual running. Therefore, we believe the transparency of the underlying cloud infrastructure is currently a general challenge for cloud application developers and has inevitably raised the application development and deployment barrier.

A notable aspect of our current pipeline is that it does not leverage an existing MapReduce framework such as Hadoop [20] or Dryad [21]. The reason for this is two-fold. First, none of these frameworks are currently supported in Windows Azure Platform. As of the time of this paper, Windows Azure has just been commercially released, and it is expected that more frameworks such as Dryad will be supported in near future. Second, we have found it "natural" to use Azure's general queue-based task model, which is quite different than a MapReduce formulation. With this loosely-coupled computation model, it will be easy for us to extend our current pipeline architecture to incorporate more complex computational and data flows. Actually we are exactly doing this now as we are adding a new stage, which is a second reduction stage to the current 3-stage pipeline. It is not clear at this time how this particular problem would/could use a MapReduce formulation. Nevertheless, we are still interested to see how our pipeline architecture could be implemented on an existing MapReduce framework. Thus, it remains to be our future work.

In conclusion, as we have demonstrated in our work, cloud computing has the great potential to eliminate or significantly lower many resource barriers and data barriers that have been existing for a long time in eScience domain. It provides a cost-effective, highly scalable and flexible solution for those large-scale, compute- and data-intensive eScience researches. At the same time, at this early stage of cloud computing, some general challenges remain to be overcome and tradeoffs need to be made by scientists and developers when consider moving their applications to the cloud.

Acknowledgments

This work is supported in part by Microsoft External Research. It is also supported in part by the Director, Office of Science, Office of Advanced Scientific Computing, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

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

[2] 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.

[3] A. Huete et al., “Overview of the radiometric and biophysical performance of the MODIS vegetation indices,” Remote Sensing of Environment, 83, 195– 213, 2002.

[4] W. E. Esaias et al., “An overview of MODIS capabilities for ocean science observations,” IEEE Trans. Geosci. Remote Sensing, vol. 36, pp. 1250–1265, July 1998.

[5] Microsoft’s Windows Azure Platform. http://www.microsoft.com/azure/default.mspx

[6] Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2/

[7] Google App Engine. http://code.google.com/appengine/

[8] Constantinos Evangelinos and Chris N. Hill, “Cloud Computing for Parallel Scientific HPC Applications: Feasbility of Running Coupled Atmosphere-Ocean Climate Models on Amazon’s EC2,” CCA-08, 2008.

[9] P. Watson, P. Lord, F. Gibson, P. Periorellis, and G. Pitsilis, “Cloud Computing for e-Science with CARMEN,” Iberian Grid Infrastructure Conference, May 2008.

[10] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, “Science Clouds: Early Experiences in Cloud Computing for Scientific Applications,” Cloud Computing and Applications, 2008.

[11] C. Hoffa, G. Mehta, T. Freeman, E. Deelman et al., “On the Use of Cloud Computing for Scientific Workflows”, IEEE Fourth International Conference on eScience (eScience 2008), Dec. 2008.

[12] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good. “The Cost of Doing Science on the Cloud: The Montage Example,” Proceeding of Super Computing 2008, November 2008.

[13] C. Vecchiola, M. Abedini, M. Kirley, X. Chu, and R. Buyya, "Classification of Gene Expression Data on Public Clouds," Technical Report, CLOUDS-TR-2009-7, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Aug. 7, 2009.

[14] Z. Hill and M. Humphrey, “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.

[15] Christian Baun, Marcel Kunze, “Performance measurement of a private Cloud in the OpenCirrus Testbed,” 4th Workshop on Virtualization in High-Performance Cloud Computing (VHPC 2009). Aug 2009.

[16] “Primary Data Search,” https://wist.echo.nasa.gov/api/

[17] MODIS Swath Reprojection Tool.
http://gcmd.nasa.gov/records/MODIS_Swath_Reprojection_Tool.html

[18] Msphinx. http://www-loa.univ-lille1.fr/Msphinx/

[19] .NET Access Control Service. http://www.microsoft.com/azure/accesscontrol.mspx

[20] “Hadoop MapReduce Framework,” http://hadoop.apache.org/mapreduce/.

[21] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: Distributed data-parallel programs from sequential building blocks,” In EuroSys 2007, March 2007.