Data-intensive science: The Terapixel and MODISAzure projects

Deb Agarwal, You-Wei Cheah, Dan Fay, Jonathan Fay, Dean Guo, Tony Hey, Marty Humphrey, Keith Jackson, Jie Li, Christophe Poulain, Youngryel Ryu and Catharine van Ingen

International Journal of High Performance Computing Applications 2011 25: 304
DOI: 10.1177/1094342011414746

THE INTERNATIONAL JOURNAL of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
25(3) 304–316
© The Author(s) 2011
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1094342011414746
hpc.sagepub.com
SAGE

Data-intensive science: The Terapixel and MODISAzure projects

Deb Agarwal[1], You-Wei Cheah[2], Dan Fay[3], Jonathan Fay[3], Dean Guo[3], Tony Hey[3], Marty Humphrey[4], Keith Jackson[1], Jie Li[4], Christophe Poulain[3], Youngryel Ryu[5], and Catharine van Ingen[3]

Abstract
We live in an era in which scientific discovery is increasingly driven by data exploration of massive datasets. Scientists today are envisioning diverse data analyses and computations that scale from the desktop to supercomputers, yet often have difficulty designing and constructing software architectures to accommodate the heterogeneous and often inconsistent data at scale. Moreover, scientific data and computational resource needs can vary widely over time. The needs grow as the science collaboration broadens or as additional data is accumulated; the computational demand can have large transients in response to seasonal field campaigns or new instrumentation breakthroughs. Cloud computing can offer a scalable, economic, on-demand model that is well matched to some of these evolving science needs. This paper presents two of our experiences over the last year – the Terapixel Project, using workflow, high-performance computing and non-structured query language data processing to render the largest astronomical image for the WorldWide Telescope, and MODISAzure, a science pipeline for image processing, deployed using the Azure Cloud infrastructure.

Keywords
cloud computing, data-intensive science, massive datasets, MODISAzure, Terapixel

1 Introduction

A thousand years ago, science was based on observation; in the last few hundred years, science has focused on using models and theories to represent reality and, for the last couple of decades, science has increasingly relied upon computation and simulation to examine complex phenomena. With the exponential growth of scientific data in almost every field of research and the convergence of mobile devices and internet-based connectivity, a fourth paradigm for research is emerging, namely that of data-intensive science (Bell et al., 2009; Hey et al., 2009). This convergence is making scientists increasingly reliant on more powerful 'eScience' tools and technologies to manage, curate, visualize and manipulate the ever-increasing amount of scientific data.

With the increasing scale and heterogeneity of datasets comes the problem of how to assemble, process, and reduce them to derive science results. There are three significant barriers.

- Resources

  Acquiring necessary computational resources to meet the requirements of these large-scale applications is difficult for many research groups. Scientists that only have remote access to high-performance clusters or supercomputing centers often experience indefinite job turnaround times due to resource sharing.

- Complexity

  As a computation scales, the coordination necessary to manage large datasets, schedule large numbers of contributing computations and support fault tolerance becomes increasingly complex. At the same time, large-scale computations are stretching the science; for environmental applications, for example, diverse ecosystems such as tropical rain forests, boreal forests, dry savannahs, and fertilized crops each present distinct challenges.

[1]Lawrence Berkeley National Lab, USA
[2]School of Informatics and Computing, Indiana University, USA
[3]Microsoft Research, USA
[4]Department of Computer Science, University of Virginia, USA
[5]Department of Organismic and Evolutionary Biology, Harvard University, USA

Corresponding author:
Tony Hey, Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
Email: tonyhey@microsoft.com

- Tedium

  Amassing and harmonizing large-scale data from diverse data sources with different spatial and temporal resolution and formats requires repeating simple tasks. Manually copying 10 files is quite possible; the tedium involved when copying 1000 or more files becomes problematic.

These barriers, if not correctly handled by software systems and appropriately masked from domain scientists, can cause invalid results in the final scientific knowledge and/or make an analysis impossibly daunting. In this paper we present two examples of the use of HPC and cloud technology in handling large-scale computations on large datasets.

2 The Terapixel project

2.1 Creating the largest, clearest seamless image of the sky

The WorldWide Telescope (WWT; see http://www.worldwidetelescope.org) is a free program from Microsoft Research that allows users to take a virtual tour of the universe from the comfort of their living room. The WWT offers the ability to seamlessly pan and zoom across a spherical view of the entire night sky in a media-rich, immersive experience that blends extensive imagery, data and stories from multiple sources on the World Wide Web. This section describes the creation of the now default sky view in the WWT – a multi-scale, seamless, high-resolution image ($2^{20} \times 2^{20}$ pixels at the most detailed level) derived from the Digitized Sky Survey (DSS) dataset (http://archive.stsci.edu/dss). To our knowledge, this image – which we call the Terapixel image because of its size – is the largest and clearest seamless image of the night sky ever produced.

Creating the Terapixel image is a multi-stage process involving input and output datasets of the order of one terabyte at every stage of the workflow. It is also computationally intensive. For tractability, parallel computations are used extensively within a machine to leverage all its cores, and across a cluster of machines. Multiple patterns of distributed computations are present in the pipeline, including embarrassingly parallel computations, tightly coupled computations and data-parallel aggregations in the style of MapReduce. Because it displays all of these characteristics, the Terapixel project is an effective demonstration of the positive ways in which modern computational methods can help facilitate scientific advances in the age of data-intensive research.

The following sections detail the process of creating the Terapixel sky view and provide an overview of the technologies that made the project successful.

2.2 Terapixel process

The Terapixel project began with data from the DSS (http://archive.stsci.edu/dss), which is a collection of thousands of photographic plates taken over five decades by two ground-based survey telescopes – the Palomar telescope in California, USA, and the UK Schmidt telescope in New South Wales, Australia. The Palomar telescope took photographs of the Northern sky, and the Southern sky down to about 30 degrees south. The UK Schmidt telescope took photographs of the rest of the Southern sky. Each photograph covers an area of six and a half square degrees. For each section of the sky, the DSS provides two separate images capturing light in the blue spectrum and the red spectrum, respectively. The photographic plates were digitized over a 15-year period. As a result, multiple scanners were used resulting in digital images that are either $14{,}000 \times 13{,}999$ or $23{,}040 \times 23{,}040$ 2-byte pixels. Terapixel used 1791 pairs of red-light and blue-light images amounting to nearly 4 TB of data in-memory. Astronomers devised an efficient compression algorithm to store the images as 28×28 or 30×30 tiles, representing 417 GB of data spread over 3,120,100 files.

Once the original files are decompressed they undergo a series of programmatic changes to correct for artifacts introduced during the telescope imaging process, such as varying levels of brightness, noise and color saturation, as well as vignetting: a darkening of the edges and the corners of each plate caused by non-uniform illumination on the telescope mirror. Next, the red and blue plates are aligned such that their corresponding pixels map to the exact same position in the sky. The aligned images are then combined to form a new color image, where the green component is the average of the red and blue components. Each color image is encoded as a portable network graphic (PNG) file and has associated metadata to map it to sky coordinates.

The next step is to stitch the color images together into a spherical image sampled using the TOAST (tessellated octahedral adaptive subdivision transform) parameterization (http://www.worldwidetelescope.org). This step produces two intermediate terapixel images stored as a 256×64 array of 8192×8912 PNG-encoded files. The first image set uses 3 bytes per pixel and is the sky view containing visible seams. The second image set with 2 bytes per pixel is a mask indicating where the seams occur.

Terapixel uses the global image optimization program developed by Hugues Hoppe and Dinoj Surendran of Microsoft Research and Michael Kazhdan of Johns Hopkins (Kazhdan et al., 2010). Gradients across image boundaries are set to zero, resulting in a seamless spherical panorama. Unlike color image creation and stitching, which involve loosely coupled distributed computations, this optimization step requires distributed processes acting in lockstep.

Figure 1 displays the sky view before and after optimization. While the output produced by the optimization is the final Terapixel image, an additional step is required to re-package the image into a multi-scale, tiled image pyramid. The tiling is necessary to provide smooth data transfers between the WWT server and its clients. The Terapixel image pyramid includes 13 levels of tiles with level n ($n = 0 \ldots 12$) containing 2^n 256×256 PNG tiles.

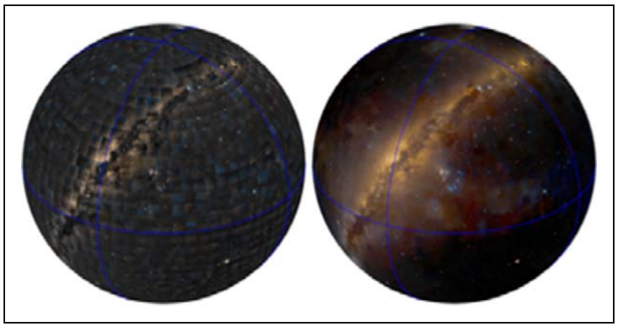

Figure 1. Before and after image smoothing.

2.3 Technical approach

Given the large amount of data and computation involved, we made use of a MapReduce-like technology from Microsoft Research called DryadLINQ (Isard et al., 2007; Yu et al., 2008), as well as the .NET parallel extensions (http://msdn.microsoft.com/en-us/library/dd460693(VS.100).aspx) to manage code running in parallel on multi-core machines of a Windows high-performance computing (HPC) cluster. We also used Trident Scientific Workflow Workbench, an open-source (Apache 2.0 license) scientific workflow design and management tool developed by Microsoft Research, to manage the overall process (http://tridentworkflow.codeplex.com). These technologies are described in more detail below.

2.3.1 Windows HPC cluster. Microsoft Windows HPC Server 2008 (http://www.microsoft.com/hpc) is the high-performance computational platform used at the core of the Terapixel project. We used a cluster consisting of 64 compute nodes, each with two quad-core Intel Xeon central processing units (CPUs), 16 GB random-access memory (RAM), and 1.7 TB of storage. This system allowed us to compute the final Terapixel from the raw digitized data in a little more than half a day: approximately 5 hours to create the color images, 3 hours to stitch them and 4¼ hours to optimize the image. Tasks are submitted to the cluster programmatically using the .NET HPC application-programming interface (API) provided by the HPC Pack 2008 software development kit (SDK) or using DryadLINQ as described below. Moving data can account for a non-negligible amount of time. For example, copying the final image pyramids ($\sim$800 GB) from the cluster to a permanent storage server over the enterprise network took nearly 2½ hours. During the overall computation, care was taken to move the computation to the data to avoid unnecessary data movements. With the existing .NET HPC API, this can be achieved by setting explicit constraints that force a task onto a specified compute node. For more complex computations, it is a lot easier to rely on a framework such as DryadLINQ.

2.3.2 DryadLINQ on a Windows HPC cluster. DryadLINQ (Isard et al., 2007; Kazhdan et al., 2010) provides an intuitive programming model for writing distributed data-parallel applications that scale from a small cluster to a large data center. To the programmer a DryadLINQ program looks like a sequential program, but it executes in a distributed fashion. This is achieved by leveraging the LINQ (Language Integrated Query) feature of the .NET framework, which provides declarative programming for data manipulation within a standard .NET language, such as C#. DryadLINQ then translates a LINQ into a distributed execution plan. The plan and necessary resources (including compiled code) are automatically sent from the client program to a Windows HPC cluster equipped with Dryad (http://connect.microsoft.com/dryadlinq). Once the distributed computation finishes, the DryadLINQ provider returns control and results to the client program.

In the Terapixel project, DryadLINQ was most usefully used to compute the correction that must be applied to each digitized plate to limit the effect of vignetting. Our approach is inspired by the procedure for vignetting correction in (Gal et al., 2004). For all images in a set of related digitized images (e.g. all blue-light plates taken from the Southern observatory), normalize each image by the median of its central region, then aggregate all the plates to form a single average plate and normalize the resulting image such that the maximum in the central region is one. Since this procedure involves hundreds of giga-pixel images, it is not straightforward to write a sequential program for a single machine. With DryadLINQ, the core of the algorithm to distribute the computation is written in a few lines:

```
var rows = images.SelectMany(image =>
ImageToRows(image, options));
var stackedRows = pixelRows.GroupBy(row =>
row.Position);
var finalRows = stackedRows.Select(x =>
ReduceStackedRows(x));
var flatfield = finalRows.Apply(x =>
SaveFlatField(x, options));
```

The programmer is left with writing sequential routines to: (1) load an image in memory, normalize it and shred it into rows of pixels (ImageToRows); (2) average values from all plates at a given pixel location (ReduceStackRows); and (3) save the aggregated results to a file (SaveFlatField). In addition, the programmer must supply the initial collection of images (images) by creating a PartitionedTable, which is the type that DryadLINQ uses to represent distributed collections of strongly typed .NET objects. A partitioned table is usually created by writing a DryadLINQ program that distributes objects onto a cluster and automatically generates the metadata necessary to describe the collection.

Generating color images is an embarrassingly parallel computation. It involves reading image files, correcting image artifacts, aligning the red and blue plates to make their corresponding pixels map to the exact same position in the sky and combining the aligned images to form a new color image. DryadLINQ was used for the entire color image creation workflow (more details later).

We did a performance comparison between a 28-node cluster and a 64-node cluster (each node with eight cores and 16 GB memory) for creating all the 1791 color images. It took 9 hours for the 28-node cluster and 5 hours for the 64-node cluster. However we believe that file inputs/outputs, including data partition and data cleanup, will become the dominant performance bottleneck if too many nodes are used for the computation. The number of cores allocated for each color image creation is also an important factor to consider. The input image tiles (blue plates and red plates) were compressed with about 10× compression factor. The first step to generate color images was to decompress all the tiles for a blue or red plate and store the decompressed plate as a two-dimensional (2D) array of short integers. We found that there was a diminishing return of performance beyond eight cores to decompress image titles on each node. From a single core to eight cores, the performance gain was about five times. However, there was very little performance gain by allocating 16 cores or 24 cores compared to eight cores. This is because file inputs and outputs became the performance bottleneck for each eight-core node. Each embarrassingly parallel computation needs to have a balance of computational resources and return of performance gains. This requires experimentation to understand the characteristics. It is highly desirable to make the computation process automated and easily configurable for reruns and performance optimization.

2.3.3 The .NET parallel extensions. The .NET parallel extensions (http://msdn.microsoft.com/en-us/library/dd460693(VS.100).aspx) refer to the new parallel programming support now built into the Microsoft .NET Framework 4 or available as an add-on to version 3.5. Many computations can be sped up by taking advantage of the multiple cores that are available on modern processors. The .NET parallel extensions provide a library of types that simplify parallel development. Many algorithms in Terapixel are easy to parallelize because they involve nested loops of the form:

```
for (int y=0; y<ny; y++)
for (int x=0; x<nx; x++)
// do work at (x,y) location
```

Since iterations are independent of each other, we can optimize performance by taking advantage of the parallel version of for:

```
Parallel.For(0, ny, y => {
for (int x=0; x<nx; x++)
// do work at (x,y) location
});
```

2.3.4 Workflow. We used the workflows created with Trident Scientific Workflow Workbench as both the front-end and back-end workflow systems. The front-end workflow running on a Windows desktop delegates the compute-intensive and data-intensive workflow to a back-end Windows HPC cluster with 64 nodes (512 cores).

Trident is layered on top of Windows Workflow Foundation (version 3.5). It is a workbench that allows one to reuse customized libraries of activities and workflows. One can visually compose workflows by connecting working activities together based on data inputs and outputs. It also allows for scaling of workflow execution from desktops to Windows HPC clusters.

We used Trident to create and manage all stages of the processing pipeline. Each stage of the Terapixel process is a Trident workflow activity, from the initial data preparation to sending the Terapixel image to the WWT. Figure 2 focuses on the step generating color plates as an example. Two workflows were used. (1) A workflow on a desktop for process automation. This workflow collects user inputs, such as the cluster name, where the raw image files are, how many HPC nodes are available for processing, etc. It stages data accordingly, by distributing data across the HPC cluster and generates the data partition files for DryadLINQ. It then uses DryadLINQ to trigger the parallel processing of data on the cluster. Note that in this case we employed DryadLINQ to distribute the computation but it could have equally used the Windows HPC .NET API since the distribution of tasks is embarrassingly parallel. (2) On the cluster side, another workflow defines the steps to generate a single color image from blue and red image data is run in parallel to generate all colored images. After the colored images are created, the client side workflow copies all data to a single share and cleans up intermediate data on each cluster node (post processing).

The use of workflows was beneficial to the project because they unambiguously documented the process at a level that is easy to comprehend. The availability of workflows also simplified the task of re-running the entire process. For example, our initial attempt at creating the Terapixel image did not include de-vignetting. It was only after visually inspecting our first image that we concluded that vignetting was not negligible. We subsequently modified the workflow and re-ran the pipeline without problems.

2.4 Terapixel conclusions

The Terapixel project is a successful demonstration of how workflow systems and HPC technologies, such as Dryad/DryadLINQ, HPC Clusters, multi-core processors and the .NET parallel extensions, create new possibilities for scientific research. A computationally intensive project, such as Terapixel, would have been difficult to achieve using conventional computing resources. The hybrid use of the workflows for data management, process automation and computational data analysis on both the local desktop workstations and remote cluster or cloud resources make

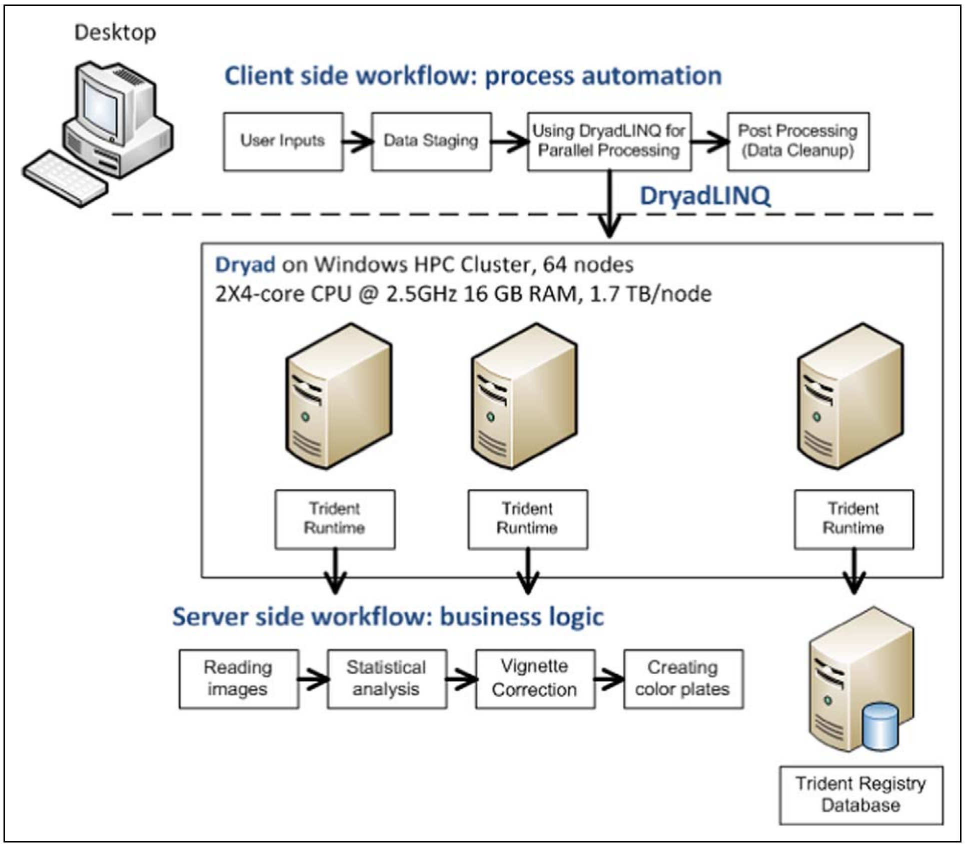
Desktop
Client side workflow: process automation
User Inputs
Data Staging
Using DryadLINQ for Parallel Processing
Post Processing (Data Cleanup)
DryadLINQ
Dryad on Windows HPC Cluster, 64 nodes
2X4-core CPU @ 2.5GHz 16 GB RAM, 1.7 TB/node
Trident Runtime
Trident Runtime
Trident Runtime
Server side workflow: business logic
Reading images
Statistical analysis
Vignette Correction
Creating color plates
Trident Registry Database

Figure 2. Terapixel deployment architecture – generation of colored images.

the workflow systems very flexible for data-intensive and computationally intensive eScience research.

3 MODISAzure: scaling science to the globe

3.1 Environmental science and the Moderate Resolution Imaging Spectroradiometer program

The confluence of remote sensing, ground-based sensors, commodity computers and internet connectivity is enabling a new era of data-intensive environmental science. At the same time, many of the environmental science questions of interest are *synthesis studies* that cross science subdisciplines and diverse data sources. The MODISAzure project (Li et al., 2010; Ryu et al., 2010; http://research.microsoft.com/en-us/projects/azure/azuremodis.aspx) addresses the computational aspects of the three barriers referred to in Section 1. The intent is to enable domain scientists to focus on the science complexity resulting from global scale diversity and subsequent data mining of the large-scale results to increase science learning.

The MODIS (Moderate Resolution Imaging Spectroradiometer) program was designed to improve the understanding of global dynamics and processes occurring on the land, in the oceans and in the lower atmosphere (Justice et al., 1998; http://modis.gsfc.nasa.gov/). The MODIS data is generated by the *Terra* and *Aqua* satellites and 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 Huete et al., 2002 for an overview with regard to vegetation and Esaias et al., 1998 for an overview with regard to ocean science). We encountered all of the three barrier issues identified above, particularly the inability to practically compute such data transformation and integration on a typical scientist's desktop machine or even a small departmental cluster.

To our knowledge, the MODISAzure project is one of the first large-scale science applications to use the Windows Azure platform (http://www.microsoft.com/azure/default.mspx). Because we built this application from scratch, we leveraged a large subset of the Azure cloud service elements as the basic building blocks for our application components. While commercial cloud computing is still in its infancy and many challenging open issues remain, we believe that the Windows Azure platform can provide a compelling approach for large-scale scientific explorations. To date, we have used more than 500,000 CPU hours, amassed an over 5 TB dataset and gained as a factor of over 100× speedup over our reference high-end scientific desktop by running on parallel Azure virtual machines (VMs). While the project was developed specifically for MODIS data processing, we believe that the experience we have gained and the lessons learned from

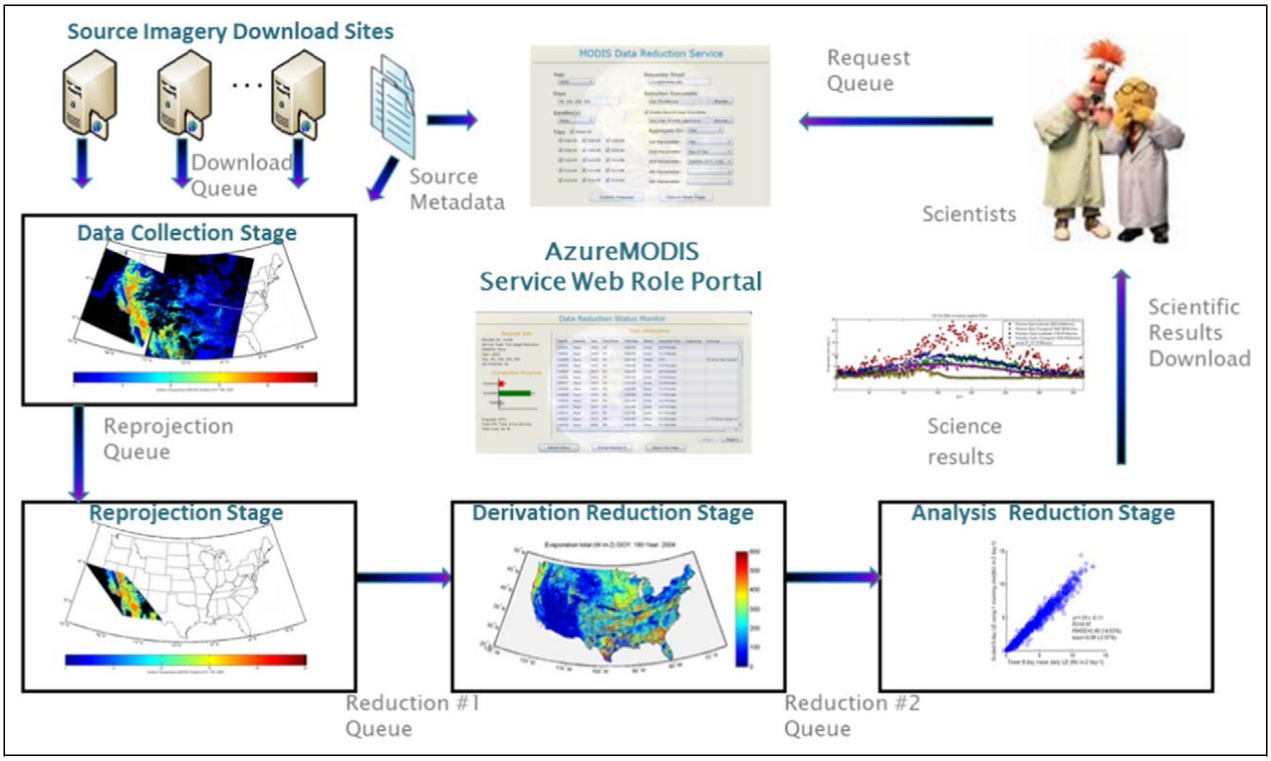

Figure 3. MODISAzure architecture.

implementing this architecture on Windows Azure can be generalized to a broad range of image-processing eScience applications.

3.2 MODISAzure

The MODISAzure project is a loosely coupled, component-based parallel data processing system running on Windows Azure as a hosted service. Windows Azure provides two types of virtualized compute instances as differentiated by *roles*:

the *Web Role* instances are Windows-based VMs hosting web applications on Internet Information Services (IIS);
the *Worker Role* instances are background Windows VMs for running customized user code.

The MODISAzure system consists of two service components running on these two different types of compute instances. As shown in Figure 3, the first service component is a front-end web portal for user job submission and execution status monitoring. This component is a Microsoft Silverlight-based web application that is hosted on a web role instance. The second component is the back-end computing system hosted on a number of worker role compute instances. It includes four main data processing stages, which can be either pipelined or run independently. Each stage performs a specific type of data-processing task as follows.

1. In the *data collection* stage, a background task running in a compute instance downloads the MODIS source data from external file transfer protocol (FTP) sites to local storage, and then uploads the data to the *blob storage*, a persistent storage service for large-scale unstructured data in Windows Azure. Prior to download, a compute instance first queries the geo-spatial information about the target source data against a source metadata table, and then goes to the specific FTP location indicated by the metadata to fetch the data. The metadata table is built using the Azure *table service*, which is the persistent storage for structured data.
2. In the *reprojection* stage, the set of heterogeneous data products collected in the first stage are reprojected into time and spatial-aligned imagery data. A set of compute-intensive algorithms (e.g. nearest-neighbor pixel match) are performed to harmonize data points pixel by pixel. The reprojected result data can then be uploaded to the blob storage for use in future scientific analysis.
3. In the *reduction* stages, a number of compute instances invoke a reduction executable uploaded by the scientist to perform the analysis computation over the reprojected data from the previous stage. The executable can be compiled from any source code, such as C/C++, MATLAB, etc. The 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 significantly smaller science analysis artifacts necessary to understand the results of the first stage.

There is also one dedicated worker role instance running as the *service monitor* (not shown), whose main responsibility is to parse science job requests into a large number of parallel tasks, send them to the task queue, and send an email to the scientist on completion. Each submitted job request specifies the relevant MODIS datasets that cover a specific date period, geographical area, scope of computation and the reduction executable(s) to be uploaded for the reduction stage computation. The service monitor parses this specification and separates the job into a large number of embarrassingly parallel tasks; each task corresponds to the scope of a single day on a single geographic unit, that is, a single *sinusoidal tile*.

The task scheduling and execution model of MODISAzure is based on the Windows Azure *queue service*, an asynchronous message-based communication service. When processing a job request, the service monitor determines the unique inputs for each task and marshals those as extensible markup language (XML) format messages and sends them to the appropriate task queue. Our Generic Worker sandbox running on the compute instance pulls task items from the queues and invokes corresponding data processing code for different types of tasks (data collection, reprojection and reduction), as well as converting each task queue message into a task table log entry. While we could have leveraged job scheduling engine such as the Microsoft HPC Server, that would not have solved the problem of converting the single science job request into the often greater than 10,000 individual imagery tasks. We would have also had to build additional tools to mine that job log to ensure that all tasks had actually succeeded and the results obtained.

We implemented a Generic Worker task execution framework similar to the one described by Simmhan et al. (2010). In this execution framework, every compute instance is capable of executing all types of tasks. In other words, we do not deploy multiple types of worker role instances in the system and then assign a specific type of task for each instance type. This execution model helps eliminate the potential load imbalance between the instances when working on different types of tasks from the queues. It is also flexible enough to support a new task type in the system without modifying the underlying service architecture. The new task processing code can be added to the framework in the form of source classes in C#, compiled libraries or executables. They will be packaged together with the service deployment to be hosted on every compute instance. In retrospect, the combination of the queue-based task dispatching and the task-polling model was the key to achieving software scalability and flexibility in our system. Instead of the task push model, there is no need for a central job scheduler in charge of managing and assigning tasks to different worker instances. Every worker instance is self-managing, and thus can dynamically enter or leave the computation resource pool. This in turn enables compute instances to be dynamically scaled up/down without impacting any of the service components, as well as implicitly load balancing work across instances. This on-demand resource scalability brought by cloud computing allows us to scale from a small regional computation up to global-level computation without any changes to the software components.

MODISAzure dynamically scales the number of compute instances for a cloud service. As discussed above, the loose-coupling and self-managing paradigm of our Generic Workers allows us to dynamically scale up/down the number of compute instances according to the real-time workloads from scientists' job requests, so as to balance the cost and responsiveness. Dynamic instance scalability can be achieved by invoking the Azure Management API to update the service configuration for a deployed application, which specifies the number of compute instances for each type of web/worker role. In MODISAzure, an independent component is deployed on the *service monitor* instance to monitor the real-time job requests submitted by scientists. When there are no job requests submitted to the queue, the service monitor reduces the number of compute instances to a minimum number to maintain service availability. Upon the submission of a new job request, the service estimates the total computational requirements for this request, calculates the number of new instances that need to be started to work on the computation, based on the criteria of turnaround time, and invokes the Management API to adjust the number of instances accordingly.

At the scale of over a quarter of a million tasks and tiles in a single job request, even rare failure events pose problems that can take significant human effort to understand and repair. A significant amount of time and effort has been devoted to identifying these failures and ensuring that the service is reliable and robust enough to automatically handle the various failures that we have faced. These failures stem from both the data scale of our application and also the characteristics of the cloud environment.

We categorize the types of failures encountered and our fault tolerance failure policies into *data failures* and *computation failures.* Data failures are caused by flaws in the data, such as corrupted data content, missing source data, etc. These errors are often domain specific, thus these failures require the scientist to incorporate fault-tolerant logic into the scientific code. Although flawed data takes up a small fraction of the datasets that we have, the consequences are severe at large scale as they may cause software failures and invalidate the results of scientific experiments. Current cloud infrastructures are built on top of commodity hardware and applications running in the cloud are prone to hardware and software failures. Computation failures are caused by transient hardware or infrastructure failures, such as slow instances or storage access,

Table 1.

	Source files	Source size	Result files	Result size
USA (10)	21,850	238 GB	27,375	261 GB
Tower (3)	80,670	993 GB	58,400	210 GB
Global (3)	152,670	2414 GB	352,225	630 GB

and are typical at the service infrastructure level. Some of the examples of these failures are slow VM instances and storage access exceptions. A typical fault-tolerant solution to overcome these failures is to implement a recovery strategy by retrying the task execution. In our service, we have implemented a customized task retry policy. For every task that times out or fails, a task is terminated and placed back in the service queue to be retried. This is performed for a certain number of retries (three times by default), before the service declares it as a failure.

Monitoring is critical for tracking and diagnosing the execution status and problems of the numerous tasks in MODISAzure. Since the number of tasks for a single job ranges from several hundred to over a quarter of a million, it is important for us to record this vast amount of information in such a way that it can be used effectively and efficiently. The Azure table service is used because it provides a structured data store that is scalable yet supports querying in an easy manner. Data from the monitoring and logging components are mainly used in one of two ways.

1. Online job execution monitoring and analysis. This is through a status-monitoring interface on the web portal that retrieves task execution information from the corresponding TaskStatus table. The execution progress and statistics for any computation task can be retrieved in real time by providing a unique job identification (ID). Other helpful information, such as the standard output and error output from the invocation of reduction executables, is also provided. From such information, scientists are able to better diagnose and debug the various problems for their executable code during the development phase. The status of each task is also tracked for fault tolerance and failed tasks are first handled by issuing a certain number of retries before finally declaring it as a failure.
2. Offline analysis. Since the table services do not provide the capability of performing complex statistical analysis over the data, we download the logged records from the tables and place them in a SQL database. By building an online analytical processing (OLAP) data cube over these data, we are able to perform richer statistical analysis across various dimensions. Comprehensive views of billing records, task status and storage consumption across time are examples of how logging records are used. Support for analyses of this kind would be impractical if implemented on the Azure tables, and we are currently evaluating SQL Azure for this mining.

3.3 Computing evapotranspiration with MODISAzure

We have been developing and operating MODISAzure since summer 2009. Our science goal was to compute evapotranspiration (ET) directly from the imagery. ET is the loss of water to the atmosphere by standing water bodies (evaporation) or plants (transpiration) and is key to understanding water balance at the regional, continental and global scales. Our computation used 17 different MODIS products; each MODIS product corresponds to a set of science variables and associated quality flags in a given projection. To handle gaps and other data issues caused by satellite outages, clouds and other issues, we also used four additional geo-spatial datasets, including global climate reanalysis products and climate classifications.

There have been four distinct phases in the project.

- Early development (7/2009–9/2009). During this phase we used a single MODIS sinusoidal data cell consisting of a raster of 1200 $\times$ 1200 1 km pixels over the year 2003.
- Continental USA scale (10/2009–3/2010). During this phase, we expanded the computation to cover the 15 sinusoidal data cells covering the continental United States for the 10 years of available data (2000–2009).
- FLUXNET tower scale (3/2010–4/2010). During this phase, we included 20 additional sinusoidal cells for the years 2003 and 2004. The time frame and sinusoidal cells were chosen to correspond to one or more of the existing FLUXNET sensor deployments so we could compare our results with the ground observations (Baldocchi, 2008; http://www.fluxdata.org).
- Global scale (5/2010–present). We expanded the computation to cover the 194 land surface sinusoidal cells for the years 2002 and 2003. We reduced the resolution to 240 $\times$ 240 5 km pixels per cell.

Table 1 summarizes the data scale of the science phases. Throughout, the MODISAzure infrastructure, reprojection algorithm and reduction algorithms evolved in response to computational scale and science data challenges.

In late fall 2009, we achieved our first one year US ET result. We then added the optional second-stage science reduction. This second stage is used to produce science analysis artifacts, such as maps, virtual sensors or plots, from the reduction computation. When reducing at scale, downloading the reduction results and then producing these artifacts on the desktop can be onerous.

In January 2010, we moved from our Azure Community Technology Preview (free, pre-release) account to a commercial account. We started monitoring our resource usage at the Microsoft Online Services Customer Portal (https://mocp.microsoftonline.com/site/default.aspx). We began to dynamically scale our deployment to keep our running costs down. We also started the practice of comparing our billed compute hours with our TaskStatus tables in February 2010; to date our

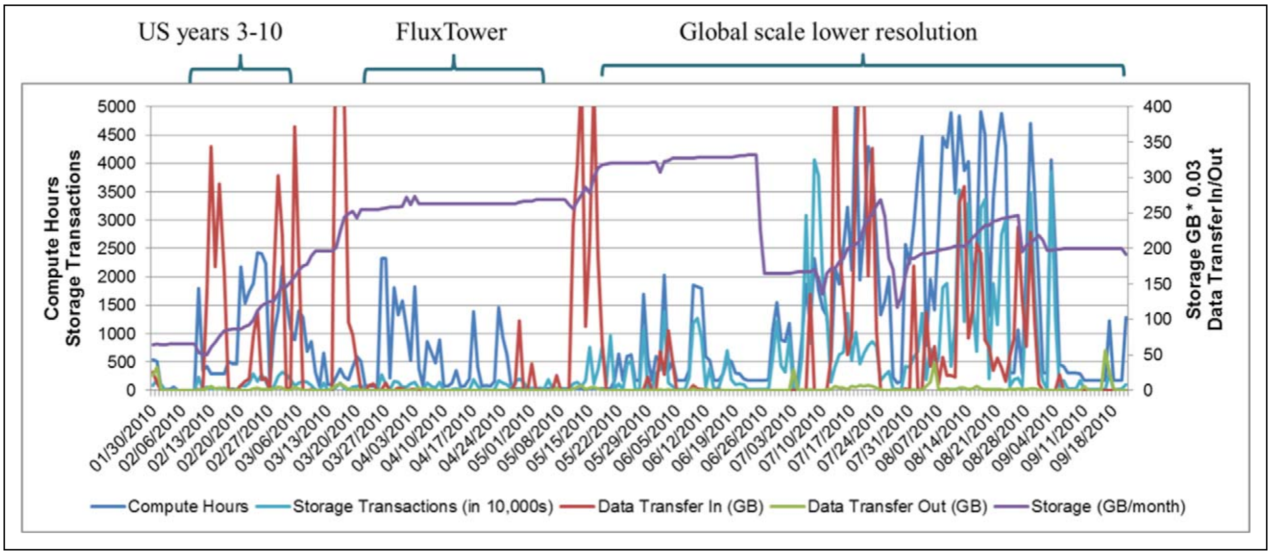

Figure 4. MODISAzure resource usage over time. The 'on-demand' nature of Azure is a good match for the changing computational requirements. At lower resolution, the global computation was almost input/output bound rather than compute bound.

task logs account for 85% of the billed compute hours. The difference is because the billed compute hours includes 'idle' instances running after task completion. We keep 'idle' instances running after task completion to avoid the need to stop/start instances unnecessarily in case of frequent job submissions while ensuring fast turnaround time for science algorithm debugging. Our billed resource consumption is shown in Figure 4.

Unlike many grid platforms, Azure billing includes upload, download and storage fees. For science convenience, most of the MODIS source tiles contain multiple science variables in a single file; we estimated that keeping only the variables needed for the ET computation would save us approximately 60% of the storage required for reprojected tiles. Since these represent over 90% of the source data, we felt this was an important saving.

We gained experience operating the service at scale in during the US computation phase. We benefitted from retrying each download, reprojection, and reduction task to reduce the impact of intermittent Azure disruptions and NASA ftp site overloads. If a task continues to fail, we use the logged status return to triage the failures and investigate to determine the nature of the failure and what, if anything, we can do to correct the problem.

Since our US computation was beginning to give good science results, we decided to expand the computation to include the additional sinusoidal tiles covering 114 additional FLUXNET eddy flux towers in late 3/2010. We expected and experienced simple scaling. Our capacity planning estimated that one FluxTower year corresponded to about 2 US satellite years (32 tiles versus 15 tiles). That resource scaling was very close – we actually consumed about 18 US satellite years. Our decision to automatically retry failed tasks served us very well in this phase; we saw approximately 6% task failure out of 57,664 tasks attempted. Of those, 41% were recovered by retry. The remaining 59% unrecoverable tasks were mainly caused by data failures or scientific code bugs.

The increased data diversity presented challenges to our ET algorithm. We encountered a much wider range of biomes, such as rainforests, and of climate regimes, such as the tropics. We found that we now needed additional science variables from the imagery; some of the layers we had previously discarded now needed to be retained.

We attempted our first global scale computation in April 2010. Based on our early success, we started the initial download and reprojection for two additional calendar years in July 2010. We chose a 5 km rather than 1 km spatial resolution based on capacity planning. The USA represents approximately 5% of the world land surface area, so we were attempting to scale the science by a factor of 20. Scaling down the resolution meant that the computation cost for reprojection and reduction of 1 US year is approximately 1 global year; the source file download and transient storage prior to reprojection are considerably higher. As seen in Figure 5, at the end of the global reprojection, we deleted all global source tile precursors for the calendar year 2003 and two extra years of FluxTower tiles. Prior to that 'storage diet', our storage bills were approximately half of our total bill.

The 5 km choice also shifted our reprojection from compute bound to slightly input/output (IO) bound. Processing each tile now took ~5 minutes of which ~2.6 minutes were spent in overhead staging tiles to/from the Azure VM instance and the blob store; a 1 km resolution tile took ~2.5 minutes of which ~0.4 minutes were staging. We also observed over 10× variation in the reprojection task time. The MODIS satellites cross a given sinusoidal tile location more often at the poles than at the equator and the number of nearest-neighbor pixels increases dramatically. We simplified the algorithm to reduce the search space across the source files and thereby reduce the overhead.

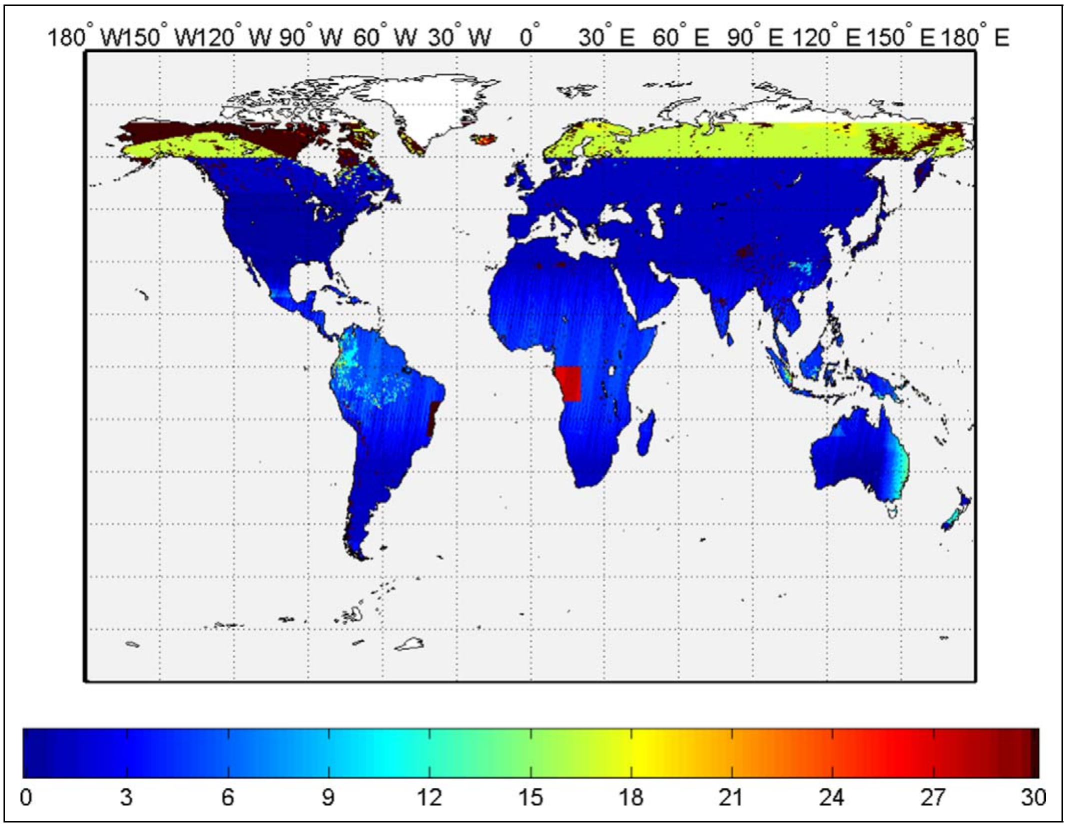
180° W150° W120° W 90° W 60° W 30° W 0° 30° E 60° E 90° E 120° E 150° E 180° E
0
3
6
9
12
15
18
21
24
27
30

Figure 5. Initial data availability for the evapotranspiration reduction in February 2004. Color coding indicates data availability; white areas were not included in the computation (color online only).

We also learned the importance of having a complete tile catalog including all source tiles on the NASA ftp sites, reprojected targets and known expected missing tiles, as well as our TaskStatus logs. Our Service Monitor experienced an Azure VM restart in the middle of scheduling the tasks for a global reprojection job with over 240 k tasks. At the same time, our download tasks were failing intermittently due to NASA site outage. Retrying both eventually generated 95% of the tiles needed for the ET reduction. We then had to track down the missing 50,000 tiles. Causes included missing tiles on the NASA site, such as on the coast of Africa, winter polar nights and (not shown) satellite outages.

Lastly, we continue to evolve the science computation and validation. Understanding how to think about regions such as the Sahara and the implications for crop fertilization remains active science research. That our pipeline is running well allows us to focus on that science.

3.4 MODISAzure summary

We provide some early observations and experiences with the development and operation of the MODISAzure application in the Windows Azure cloud computing platform. Unlike the approach used by many other eScience applications that directly move existing codes and software stacks into the cloud, we build the application from scratch on top of the basic service elements and scalable infrastructures of cloud computing.

Our decision to build a satellite image-processing pipeline leveraging the native capabilities of Azure has served us well. As we have scaled the application from Continental US to global scale, our initial service architecture has had only minor changes. We have leveraged blob service to store and manage large amounts of science data; the queue service for task dispatching and scheduling; the table service to monitor the execution status in real time and keep history logs; and the Management API to dynamically scale up/down the instances to be adapted to the dynamic workloads. Our ability to monitor and mine our JobStatus and TaskStatus tables has helped us anticipate and often simply understand our application scaling.

Our decision to 'bake in the faults' has also served us well. While Azure presents a highly reliable platform and masks many faults, our scale is such that even 99.999% reliability still creates too many faults for human examination. At the same time, the virtualized nature of Azure presents new faults such as VM substitution. Our application is delightfully parallel and the image tile is an obvious idempotent building block. This enabled us to rapidly understand how and where to build in fault retries that isolate our science user.

Lastly, our decision to use Azure tables as a common logging mechanism has given us two very important abilities. Firstly, we can monitor our application and use the accumulated measurements to plan forward. Secondly, that same forensics also gives us the ability to debug the science application code forensically.

Overall, we think cloud computing has provided an appealing environment for building scalable, data-intensive eScience applications. However, in this early stage, it still has some limitations on the application development and execution processes. For example, we have to pay a lot of attention to fault tolerance during application development, and sometimes we do not have enough control over the underlying infrastructure in order to optimize for performance. The hosted environment and black-box nature of cloud computing indicate that we will at least have to live with that for a long time.

4 Concluding remarks

These examples have shown how powerful new tools, such as generalized MapReduce frameworks like Dryad, parallel languages and libraries (DryadLINQ and the .NET parallel toolkit), workflow management, such as Trident, can be used together with HPC and cloud computing resources to effectively perform large-scale computations over large datasets. At present, the use of such technologies in scientific analysis is only just beginning: we believe that such tools and technologies will become increasingly common as the data deluge continues to grow.

Funding

The research and development described in this paper is funded by the Microsoft Corporation. This work is also supported by the Director, Office of Science, of the US Department of Energy (contract number DE-AC02-05CH11231).

Conflict of interest statement

None declared.

References

Baldocchi DD (2008) ‘Breathing’ of the terrestrial biosphere: lessons learned from a global network of carbon dioxide flux measurement systems. *Austral J Botany* 56: 1-26.

Bell G, Hey T and Szalay A (2009) Beyond the data deluge. *Science* 323(5919):1297-1298.

Esaias WE, Abbott MR, Barton I, Brown OB, Campbell JW, Carder KL, et al. (1998) An overview of MODIS capabilities for ocean science observations. *IEEE Trans Geosci Remote Sensing* 36: 1250–1265.

Gal RR, De Carvalho RR, Odewahn SC, Djorgovski SG, Mahabal A, Brunner RJ, et al. (2004) The Digitized Second Palomar Observatory Sky Survey (DPOSS). II. Photometric calibration. *Astronom J* 128: 3082-3091.

Hey T, Tansley S and Tolle K (2009) *The Fourth Paradigm: Data-intensive Scientific Discovery*. Microsoft Research. Available at: http://research.microsoft.com/fourthparadigm.

Huete A, Didan K, Miura T, Rodriguez EP, Gao X and Ferreira LG (2002) Overview of the radiometric and biophysical performance of the MODIS vegetation indices. *Remote Sens Environ* 83: 195–213.

Isard M, Budiu M, Yu Y, Birrell A and Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. In: *Proceedings of the European Conference on Computer Systems (EuroSys)*, Lisbon, Portugal, 21–23 March.

Justice C, et al. (1998) The Moderate Resolution Imaging Spectroradiometer (MODIS): land remote sensing for global change research. *IEEE Trans Geosci Remote Sens* 36: 1313-1323.

Kazhdan M, Surendran D and Hoppe H (2010) Distributed gradient-domain processing of planar and spherical images. *ACM Trans Graphics* 29: 14.

Li J, Agarwal D, Humphrey M, van Ingen C, Jackson K and Ryu Y (2010) eScience in the cloud: a MODIS satellite data reprojection and reduction pipeline in the windows azure platform. In: *Proceedings of the International Parallel & Distributed Processing Symposium*, IEEE.

Ryu Y, Li J, van Ingen C, Agarwal D, Jackson K, Cheah Y-W, et al. (2010) Global remote sensing in a PC: cloud computing as a new tool to scale land surface fluxes from plot to the globe. *Fluxletter* 3: 9-13.

SimmhanY, van Ingen C, Subramanian G and Li J (2010) Bridging the gap between desktop and the cloud for eScience applications. In: *Proceedings of the 3rd International Conference on Cloud Computing*.

Yu Y, Isard M, Fetterly D, Budiu M, Erlingsson U, Kumar Gunda P, et al. (2008) DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In: *Proceedings of the Symposium on Operating System Design and Implementation (OSDI)*, San Diego, CA, 8–10 December.

Author’s Biographies

Dr. Deborah Agarwal is a Staff Scientist at Lawrence Berkeley National Laboratory(LBNL) and the Advanced Computing for Science Department (http://acs.lbl.gov), Head and the Data Intensive Systems Group Lead at Lawrence Berkeley National Laboratory. Dr. Agarwal’s current research focuses on developing tools to enable data synthesis and improve data usability, accessibility, and sharing in science. Her group is host to the National Soil Carbon Network, the global FLUXNET database, and a California Hydrologic Information Server. Dr. Agarwal earned her BS in Mechanical Engineering from Purdue University. Her MS and PhD are from University of California, Santa Barbara in Computer Engineering.

You-Wei Cheah is a PhD student at the Indiana University School of Informatics and Computing in Bloomington, working under the supervision of Professor Beth Plale. His general research interests include distributed systems, data provenance, and large-scale data analysis.

Daniel Fay is the Director of Earth, Energy, and Environment for Microsoft Research Connections, Microsoft Corporation, where he works with academic research projects focused on utilizing computing technologies to aid in scientific and engineering research. Dan has project experience working with High Performance Computing, Grid Computing, collaboration and visualization tools in scientific research. Dan was previously the manager of eScience Program in Microsoft Research where he started Microsoft's engagements in eScience including the MSR eScience workshop.

Jonathan Fay is a Principal Software Architect at Microsoft Research leading the development of the WWT project. He has been employed at Microsoft for 18 years and shipped product as either a development manager or architect spanning photo sharing, image processing, mass-scale internet commerce and website hosting services. His interest in astronomy long predates his work on the WWT, having creating packages for astronomical images and building an automated domed observatory from scratch.

Dean Guo is a Principal Program Manager at Microsoft Research. He has a PhD in Bioinformatics from the University of Utah. He has been working at Microsoft for 10 years. At Microsoft Research in the last three years, he has been focusing on developing state of the art research tools for researchers. He was a core team member for Project Trident: A Scientific Workflow Workbench and the TeraPixel project - creating the largest and clearest image of the night sky. He is a core team member for other research projects, such as location-based data visualization using Excel and WorldWide Telescope and Try F# (running F# interactively in a browser).

Tony Hey is the corporate vice president in Microsoft Research. He is responsible for worldwide university research collaborations with Microsoft researchers. Hey is also responsible the multidisciplinary eScience Research Group within Microsoft Research. Before joining Microsoft, Hey served as director of the U.K.'s e-Science Initiative, managing the government's efforts to build a new scientific infrastructure for collaborative, multidisciplinary, data-intensive research projects. Before leading this initiative, Hey led a research group in the area of parallel computing and was Head of the School of Electronics and Computer Science, and Dean of Engineering and Applied Science at the University of Southampton. Hey is a fellow of the U.K.'s Royal Academy of Engineering and was awarded a CBE for services to science in 2005. He is also a fellow of the British Computer Society, the Institute of Engineering and Technology, the Institute of Physics, and the U.S. American Association for the Advancement of Science (AAAS). Tony Hey has written books on particle physics and computing and has a passionate interest in communicating the excitement of science and technology to young people. He has co-authored "popular" books on quantum mechanics and on relativity.

Marty Humphrey is an Associate Professor in the Department of Computer Science at the University of Virginia. He received a B.S. and M.S. degree in Electrical Engineering from Clarkson University in 1986 and 1989, respectively. He received his Ph.D. degree in computer science from the University of Massachusetts in 1996. From 1996-1998, he was an Assistant Professor of Computer Science and Engineering at the University of Colorado at Denver. From 1998-2002, he was a Research Assistant Professor at UVa. From 2002-2008, he was an Assistant Professor at UVa.

Keith Jackson is currently a Scientist at the Lawrence Berkeley National Laboratory, where he is a member of the Data Intensive Systems Group. He is currently researching methods for utilizing cloud computing for scientific applications. He is also researching methods for data-intensive scientific computing. He has been involved extensively in building component based interfaces to distributed systems, and prototyping large-scale computational and data Grids. He has been involved in developing Python interfaces to the Globus Tookit® (pyGlobus), a PKI based authorization system (Akenti), and a secure advanced reservation system (STARS). His interests include distributed systems, distributed system security, advanced reservations, network quality of service, component based middleware, and PKI based applications.

Jie Li received the B.S. degree in electrical engineering from Tsinghua University, Beijing, China, in 2006, and the M.S. degree in computer engineering from the University of Virginia in 2008. He is currently a Ph.D. student in computer engineering at University of Virginia. His current research interests include Internetworking Protocols and next generation Internet.

Christophe Poulain is a senior research software design engineer at Microsoft Research. His fifteen year experience in commercial software development has focused on building state-of-the-art systems to solve problems in science and engineering, in particular in the areas of process simulation, real-time large-scale optimization and data-intensive computations. He holds a Ph.D. in Chemical Engineering.

Youngryel Ryu received PhD at UC Berkeley in 2010. He did a research internship at Microsoft Research in 2010 summer, and conducted a postdoctoral research at Harvard University until 2011 spring. Currently, he is an assistant professor at Seoul National University, South Korea. His main research interest is understanding and predicting interactions between biosphere and atmosphere, and he has enthusiastically collaborated with computer scientists to process huge, complex, inhomogenous environmental data to answer scientific questions.

Catharine van Ingen is a Partner Architect in the Microsoft Research eScience Group. Her research centers on the use of commercial technologies for assembling, curating, and mining environmental datasets from the very large (PB and TB at the PanStarrs Telescope) to the very small (tree rings used in carbon balance estimates). She holds a PhD in Civil Engineering from California Institute of Technology.