Abstract—Environmental data arriving constantly from satellites and weather stations are used to compute weather coefficients that are essential for agriculture and viticulture. For example, the reference evapotranspiration ($ET_0$) coefficient, overlaid on regional maps, is provided each day by the California Department of Water Resources to local farmers and turf managers to plan daily water use. Scaling out single-processor compute/data intensive applications operating on real-time data to support more users and higher-resolution data poses data engineering challenges. Cloud computing helps data providers expand resource capacity to meet growing needs besides supporting scientific needs like reprocessing historic data using new models. In this article, we examine migration of a legacy script used for daily $ET_0$ computation by CIMIS to a workflow model that eases deployment to and scaling on the Windows Azure Cloud. Our architecture incorporates a direct streaming model into Cloud virtual machines (VMs) that improves the performance by 130% to 160% for our workflow over using Cloud storage for data staging, used commonly. The streaming workflows achieve runtimes comparable to desktop execution for single VMs and a linear speed-up when using multiple VMs, thus allowing computation of environmental coefficients at a much larger resolution than done presently.

I. INTRODUCTION

The growing prevalence of instruments and sensors deployed to continuously observe nature at fine spatial and temporal granularity is causing a surge in the volume of observational data collected, leading to increased resource needs for storing, managing, analyzing and sharing it [13]. This is particularly true for environmental data that stream in from satellites like MODIS and GOES [9, 21], ground instruments like RADARs and weather stations, and sensors erected in forest towers and streambeds [1]. Much of environmental sciences data is collected and managed at the long-tail of scientific research using monolithic, legacy tools running on single-system servers serving a small community of users [11, 31]. As the rate of observational data increases due to instrument upgrades and more researchers demand access to these data silos to unlock their value for multi-disciplinary sciences, data managers are looking to modularize and manage their data processing tools, and expand to scalable compute and data resources to support larger communities and more complex analysis [20].

Scientific workflows have proven successful in several scientific domains for modeling in silico experiments and managing scientific data for collaborations [35]. They provide easy composition of complex data processing and computational tasks into control and/or data flows that can be enacted on distributed resources. Workflow systems like Taverna [30], Kepler [19], and Trident [32] all provide features for visual composition and orchestration to support data-intensive applications such as environmental sciences.

Cloud computing is emerging as a viable platform for executing these scientific workflows and/or their constituent tasks [17, 37]. Clouds hold the promise of on-demand availability of compute and data resources using simple service abstractions and a pay as you go scheme. Cloud resources are more accessible to a large fraction of scientific users, when compared to costly capital and management costs of local clusters or the complexity and access limitations of HPC centers [3]. Workflow systems are increasingly supporting various public and private Clouds as execution environments. These features make it attractive for environmental data managers to migrate their data management and analysis tools to workflows that run applications in the Cloud to scale out over time.

However, there are technical and data engineering challenges posed when migrating existing legacy application and scripts to workflows and Clouds, key among them being the time taken to move data between the desktop data source and public Cloud resources where the data analysis and storage takes place. Typically, workflows transfer data from desktop to Cloud storage like Amazon S3 [2] or Azure Blob store [25] for staging and then download them to the Cloud Virtual Machines (VMs) for local execution. Workflows that execute data-intensive tasks across desktop and Cloud resources need to contend with lower bandwidth speeds in and out of Cloud storage as compared to Gbit networks available on LAN and academic WAN. This bottleneck is particularly punitive when running time-sensitive workflows operating on observational data that continuously stream in from instruments. In addition, the data rate to Cloud storage services is non-homogeneous and depends on the current load on the Cloud fabric [4]. The reliable, persistent storage provided by them using multiple copies of a file also increases the write overhead as compared to a network file system.

In this paper, we address this issue by using a streaming model to directly push and pull data from the desktop to Cloud
VMs using network sockets, without going through the slower data storage services as an intermediary staging location. We observe that direct network connection to the VM provides a better and more consistent bandwidth than from desktop to public Cloud storage. By introducing data streams as first class data members, our desktop workflow tasks can communicate with their component application logic running in the Cloud using high-performant streaming while keeping the streaming complexity transparent to users. The trade-off is the transient nature of VMs and their local disks that can potentially cause data loss. This is mitigated by checkpointing data to persistent Cloud storage at appropriate points in the workflow.

Support for data streaming in workflows is a nascent area of research. We show that in addition to making it intuitive for programming applications handling streaming observational data and the inherent pipeline parallelism offered, this feature also provides better performance for data intensive workflow applications running in the Cloud.

We make the following two contributions in this paper:

1) We describe our experience migrating and scaling a data-intensive environmental science application, used by the California Irrigation Management Information System (CIMIS) to provide daily evapotranspiration ($ET_0$) analysis to farmers, from a scripting model running on a single Linux workstation to a scalable workflow model running across Linux desktop and Windows Azure Cloud; and

2) We implement and empirically evaluate a streaming model for data transfer between desktop workflow tasks and application logic running in the Cloud, which achieves a 130% improvement of total workflow runtime over using the Cloud storage service.

The rest of the paper is organized as follows: Section II introduces the CIMIS application for calculating daily evapotranspiration ($ET_0$); Section III describes our RestFlow workflow system and its integration with Windows Azure for Cloud execution; Section IV describes the migration of the $ET_0$ application to RestFlow Cloud workflows and the improved design of using data streaming; Section V empirically evaluates the performance and scalability of the $ET_0$ workflow against its standalone scripting version, and demonstrates the benefits of streaming mode over Cloud storage transfers; Section VI provides related work, and Section VII presents our conclusions.

II. Application Background

The Geostationary Operational Environmental Satellite (GOES) provides real-time observations of spectral bands that can inform weather related parameters like the cloud coverage at a certain location [9]. However, these raw data feeds need further processing to higher order coefficients, such as relative cloud coverage (CC) and daily evapotranspiration ($ET_0$), before they can be ingested and integrated with environmental variables from other sources for meaningful analysis and visualization. CC is computed by comparing the current visible brightness to the unclouded surface albedo (reflectivity). $ET_0$ is derived from CC data and additional sensor data [10]. $ET_0$ maps are particularly important for defining water resource policy and for resource managers to develop water budgets that improve irrigation scheduling and monitor water stress. To this end, the CIMIS computes and provides daily $ET_0$ maps online [7, 38] that local Californian farmers and vineyards depend on to plan their daily plant irrigation.

Algorithms for calculating CC and $ET_0$ are compute and data intensive, and operate over a temporal moving window on data streaming in from satellite observations and historic data.

Currently, CIMIS generates $ET_0$ maps only once per day and for the California region at a spatial resolution of 2x2 km. This uses ca. 315 MB of accumulated input data from satellites and 64 MB of slow changing, historic data to generates 500MB of daily raster-images data. The CC -> $ET_0$ pipeline takes about 12 minutes to run on a dedicated dual-core, 2.8GHz server at CIMIS.

Ideally, the $ET_0$ map needs to be computed once every daylight hour, include neighboring geographic regions (including Oregon and Washington), and use a much higher spatial resolution of 8x8 m for which data is available to accurately consider local effects. This magnifies the compute and data problem by a factor of 5000, which is beyond the capacity of local CIMIS resources.

In addition, the current application pipeline consists of a series of complex makefiles that call perl scripts and binaries, and designed monolithically with little flexibility to add new algorithms, modify data sources or share with other researchers. Also, the scripts are designed to run in batch mode over satellite observation that are accumulated into files, rather than operate on streaming data as it is available from instruments and generating derived products for consumers with low latency that makes it more valuable. The management of the daily runs and programming complexity in changing their behavior add to the compute, data and algorithmic complexities inherent in the application.

These motivate the need for frameworks and programming abstractions like scientific workflows to better manage the computation and dataflow pipeline for regular execution in production mode, for sharing with external researchers to run on demand with new datasets and science variables, and to perform large scale reprocessing of historic data using novel algorithms. This also opens the door for Cloud computing, given the paucity of local resources and expertise to manage cluster infrastructure. An added incentive is easier sharing of compute and data with collaborators. Support for streaming data using a workflow abstraction intuitively matches the programming model with the data flow between observational data and low latency output data products.

III. Workflow System Design

Our system consists of a workflow engine that runs on the desktop client, with the compute and data intensive tasks that constitute the workflow being executed in the Cloud. We use RestFlow, based on the Java Spring Framework, as
our workflow engine [23, 24]. RestFlow combines the rapid prototyping advantage of scripting languages with the benefits of using a dataflow-oriented workflow system. It provides a data-oriented model of computation with task abstractions and automated tracing of invocations and data flow. For the Cloud platform, we choose Microsoft Windows Azure, which provides virtual machines called Workers running Windows Server 2008 for user computation and scalable queues, tables and blob files for persistent storage that are accessed using a REST service API [25]. Azure provides features of both Infrastructure as a Service (IaaS) Clouds like Amazon EC2 as well as Platform as a Service (PaaS) based on the .NET framework [36]. An Azure Worker is essentially a .NET program that encodes application logic and provides the entry point into a virtual machine instance. Worker VMs and data storage services can be colocated within the same data center.

A. RestFlow Workflows on Azure Cloud

The Azure Worker VMs wrap domain applications and implement common data management operations. Tasks in the RestFlow workflow interact with Workers using Azure Queues that act as a reliable message bus carrying XML-encoded Request and Response messages between client workflow tasks and the Workers. Typically, multiple Worker instances listen on a shared queue for Request messages and the first worker to dequeue the message works on this. This ensures simple load balancing among active VM instances. The result of a Request is put as a Response message on a unique queue created for that Request that the client listens on.

Our workflows are stateful, so sequences of tasks in the same workflow share common state beyond what is present in their input and output parameters. Retaining the state completely in the client-side workflow requires costly data movement between Worker VMs holding partial state locally and persistent storage in the Cloud storage or the workflow client. For example, a file created by a Worker VM on its local disk as part of a workflow task may be used as input by a subsequent task. However, there is no guarantee that the same Worker instance will receive the Request message from the next workflow task for the local file path to be valid.

To address this, the first task our workflows perform is to “reserve” a Worker instance that stops polling the shared queue and switches to a new Request/Response queue created just for that workflow instance. The last task of the workflow is to “release” the reserved Worker instance so it goes back to polling the shared queue. This model guarantees progress of the workflow once a worker has been successfully reserved, but has the trade-off of requiring an entire VM to be exclusively assigned to the workflow till it completes. We will revisit this “batch” model in future as we have more workflow instances than available VM instances and we have to perform fair load balancing that ensure progress of all workflows.

IV. Migrating Local Computation of ET₀ to a Cloud-enabled Workflow

The conceptual cloud-enabled workflow (Fig. 1) consists of three phases: (1) move satellite data from desktop to Cloud, (2) run domain computation to generate the ET₀ images, and (3) publish the results to a web server for online access and download, and clean the local VM storage for the next workflow instance.

In Phase (1), we upload the workflow input data to Azure BLOB store, reserve an Azure worker VM, and instruct the worker VM to download the input data from BLOB-store to local disk. Input data comprises of 14 days of GOES data imagery and CIMIS weather station data that is typically around 315 MB in size.

After all data has been staged, Phase (2) computes the cloud-cover for each GOES raster image available on the most recent day uploaded followed by the evapotranspiration ET₀ of this day. Computing both cloud-cover and ET₀ is a complex domain-specific task for which we reuse the leaf scripts of the existing local implementation, orchestrating the higher-level tasks from the workflow. The leaf processes are launched by a complex (975 line) Makefile which forms the application logic for the workflow tasks. To compute ET₀ for one day, 1056 targets and dependencies are specified inside the Makefile. The make rules call GIS GRASS computations [27] and Perl scripts. We used the Generic Worker windows library [33], Strawberry Perl, together with GRASS 6.4 built on OSGeo4W as the windows VM execution environment. Though cloud-cover and ET₀ execute in different tasks, they share intermediate data present in the local VM disk. Since the same worker VM instance is used by both these tasks (and the entire workflow), they can share these files without having to go though an external storage service. As a last step in the make script, we copy the computed cloud-cover and evapotranspiration maps to a designated output directory inside the local VM disk.

Phase (3) uploads the output map data products to persistent BLOB storage and returns a URL for download and for browsing the generated maps from the desktop. Azure web-hosting worker VMs serve these generated maps to support a large number of users. As a last workflow task, we call the clean target of the Makefile to remove intermediary data products from the VM and release the worker VM to be used by subsequent workflows.

Fig. 1. Conceptual workflow for computing Cloud-Cover and ET₀ in the Cloud.

1There were 1209 targets in the Make-database (created by make -qp cloud-cover et0), minus 153 default targets built into Make.
Fig. 2 shows the RestFlow workflow (left) and the communication between desktop, Azure VM and BLOB storage. The RestFlow workflow running on the desktop interacts via the REST storage API with the Azure Cloud storage resources (Queues and BLOBs). RestFlow tasks use a custom Java library to access the Azure storage services using the Azure Java storage client SDK [40].

**A. Improved Data Movement using Streams**

BLOB storage provides a convenient abstraction for moving data in and out of the cloud through a simple REST API. It is similar to shared scratch space used in HPC centers when running applications on clusters. However, we observe BLOB storage to provide a slower bandwidth from desktop to the Cloud, possibly due to data persistence guaranteed through 3 distributed copies of each BLOB. We propose an alternate design where we replace the two-step BLOB upload from the desktop client and BLOB download from the Azure VM with a direct data streaming from the desktop to the VM via a TCP socket. While the Azure fabric does not expose public IPs to every worker VM instance, it provides a network IP to a shared load-balancer for the worker deployments. This redirects a socket connection to one of the worker VMs in the deployment that is selected by an unpublished balancing strategy.

Using this model, the first task in the $ET_0$ workflow is replaced by the streaming data model while the remaining tasks of the workflow (i.e., phases 2 and 3) are not altered (see Fig. 3). The selection of a worker VM by the load-balancer also acts as an implicit Reserve worker task. We still communicate individual workflow steps from the Client- side RestFlow workflow to the Azure worker VM via Queues, and use BLOB storage to publish the final output data for persistence after the worker VM has been released.

**V. Evaluation**

We investigate the effectiveness of the cloud implementation by comparing runtimes of the two variants of the cloud workflow implementations (i.e., the BLOB and the streaming model) with the runtime achieved on the current local production environment. Besides single (virtual) machine comparison, we also demonstrate scale-out speed-up achieved using concurrent VM instances.

**A. Experimental Setup**

The current production machine at CIMIS is a dual-core 2.8GHz with 2 GB RAM, running Debian Linux and hosted on the UC Davis network. The home and storage directories for all computed data products are NFS-mounted from a storage server connected via gigabit ethernet, achieving a disk bandwidth of ca. 20 MB/sec$^2$.

As ‘desktop’ machine to run the RestFlow workflow, we used a dual Quad-Core Opteron 2.8GHz server with 32 GB RAM running Debian Linux. The CPU power of this machine is less significant since all application logic is in the Cloud and the desktop only performs data zip and transfer. We chose this machine due to its gigabit uplink to the internet via the UC Davis Genome Center. We use small Azure worker VMs rated at 1.6GHz with 1.7GB RAM and 250GB local disk space, running 64bit Windows Server 2008 in user mode. The Azure worker VMs and the data storage accounts were collocated in the same US North Central data center. We enable the extended trace functionality of the RestFlow engine to measure the actor invocation times.

All experiments were performed 4 times and their averages reported below.

**B. Single Local and Cloud Executions**

This experiment compares running a single instance of the application on the CIMIS production machine and a single

\[ \text{ET}_0 \text{ Workflow using Streaming Strategy} \]
workflows show near constant time to run in parallel while the time for running workflows locally increases linearly.

C. Speedup of Concurrent Cloud Workflows

One of the key advantages offered by Cloud computing is the seemingly unlimited resources to scale-out applications. To evaluate the practical speed-up offered by running concurrent workflows on independent cloud VMs, we ran 7 workflows to calculate ET₀ for different days (each operating over 14 days of input data of about 315MB each). Fig. 5 compares the total runtimes for calculating ET₀ for the 7 days sequentially on the production machine with running 7 concurrent workflows on 7 Azure worker instances. The total time for running a single day ET₀ workflow (discussed earlier) is also shown on the left for reference.

We observed that the computational tasks (make cloud-cover and make et0) show linear speed-up, as seen in consistent times for ‘Make Cloud Cover’ and ‘Make ET0’ in Fig. 6. In fact, the average time taken for ‘make ET0’ task was faster by 53secs in the concurrent streaming workflow runs than the single workflow run. Given that the workers run on virtualized resources whose physical hardware are shared with multiple users, this can be attributed to skewed load on the worker VM instance running the single workflow tests. Data transfers in both the Cloud workflows show an overhead of 25% to the storage service (for BLOB workflows) and load-balancer endpoint (streaming workflow) in the concurrent tests as compared to a single workflow run. For example, input data streaming takes about 220secs for the 7 workflows compared to an average of 177secs for the single workflow. This indicates that network data transfers in both BLOB and streaming modes may scale sub-linear while compute and I/O intensive tasks will scale linearly. We also saw additional overhead when performing 7 concurrent Zip tasks on the local desktop by the workflows in the concurrent test due to the high local disk I/O. With these sources of variations, the speedup observed for the concurrent workflows was a factor of 0.80 for BLOB workflows and 0.92 for streaming workflow, which is close to an ideal linear speedup of 1.00. In our future work, we will experiment with larger scale-out and from multiple desktop machines running the workflow engine to investigate possible bandwidth limitations imposed by the Windows Azure fabric.

D. Comparison of BLOB and Streaming for Data Transfer

We first investigate BLOB and streaming mode for data transfers when running a single workflow on one worker VM using input data sizes of 315MB. Transferring this data present in 14 zip files from the desktop to Cloud BLOB storage (single thread, single connection) took an average of 1030 secs (ranging from 805s to 1178s on 4 different runs, with σ=13%). This corresponds to a transfer bandwidth of about 300KB/s. Downloading data from BLOB storage to the worker VM was fast, taking about 32secs with a speed of 10MB/s. However, in one instance we observed an outlier download of 600 seconds. This is another indication that despite the promise of high availability and vast scale-out of public Cloud resources, their consistent quality of service is not guaranteed.

In the streaming mode, we could repeatedly observe better data transfer times. Streaming data from the local desktop to the cloud VM took an average of 180 seconds (ranging from 800s to 1821s on 7x4 different runs, with σ=6%). This confirms our expectation that streaming data directly to the worker VM achieves better performance than going through the persistent Cloud BLOB storage. Our initial data also suggests that network transfer to the VM is more stable than through BLOB storage (6% vs. 13% standard derivation). When performing the same test on 7 concurrent workflows running on 7 different worker instances (Fig. 6), data movement to the BLOB store takes an average 1172 seconds (ranging from 800s to 1021s on 7x4 different runs, with σ=44%). Thus, concurrent BLOB upload speeds vary between 175KB/s and 390KB/s with an average of

![Graph showing speedup of concurrent workflows over single workflow. Total time for running 1 workflow on 1 workstation/VM and 7 workflows on 1 workstation/7 VMs are shown. Cloud workflows show near constant time to run in parallel while the time for running workflows locally increases linearly.](image)
Fig. 6. Time taken for each task in workflow by each of the 7 concurrent workflows. Uniform time is observed for most tasks on different VMs, except Blob transfer time. Each bar corresponds to an average value obtained from 4 runs. BLOB Strategy is shown on the left, Streaming on the right.

270KBytes/s. Downloading from BLOB to the VM is comparable to the single workflow case with an average of 31 seconds or 10MB/s. It too exhibits a significant standard deviation of 35%.

The concurrent streaming model yet again shows better performance and consistency. The average time for streaming the data into the cloud was 227secs (1.35MB/s) for the 7 workflows. We observe that the streaming time only varies marginally (with a range of 194secs to 250secs on 7x4 different runs, with $\sigma=6\%$).

To summarize, data transfer in the streaming model outperforms transfer via BLOB storage by more than a factor of 5 on average with superior bandwidth uniformity. This results in performance improvements of 130% and 160% for the total workflow execution time for single and concurrent workload, respectively (Fig. 5).

VI. RELATED WORK

Many scientific workflow systems support modeling of data and compute intensive applications [8, 19, 32, 34]. These can be executed on local or distributed resources such as clusters, HPC centers and, more recently, the Cloud [6, 15, 17]. These workflows use virtual machines and data storage offered by public Cloud providers [2, 39] and private Cloud installations [18, 28] to execute the workflows. Often, the execution model on the Cloud is a straight port from the non-Cloud execution with updated interfaces. There is still ongoing research in identifying the unique challenges posed by Cloud resources, such as the one we have identified.

There have been several studies examining the characteristics of public Cloud resources [14, 16]. We use insights from these and our own experiences to circumvent challenges (such as low bandwidth to storage service) and take advantage of Cloud features (like direct socket connection to VMs) to provide better performance to Cloud users.

Data streaming is commonly used for sensor networks and managing instrument data [5, 26]. Standards such as OGSA-DAI provide open protocols for streaming data [29] and is used for some scientific applications [22]. There is also emerging interest in using streams for high-performance data flow within scientific workflows [12]. To our knowledge, there has not been active work on using data streams with Cloud computing resources in the context of scientific workflows.

VII. CONCLUSION AND FUTURE WORK

We presented our initial efforts in migrating a monolithic eScience application to a flexible, and scalable cloud platform. By integrating Scientific Workflow technology on the client-side, we were able to provide a high-level abstraction of the computation pipeline that makes it more accessible to a wider audience of researchers.

For single application runs, our workflow model using BLOBs were slower than the existing production scripts. However, the alternative communication model using direct streaming from client to cloud performed as well or better than the current scripts. For concurrent workloads, we observed almost linear scaling for computation and data transfer, validating the use of Cloud platforms for increased access to resources. Streaming data from desktop to cloud was on average 5 times faster than the conventional BLOB storage transfer solution, with more stable communication times. Using the streaming approach, our workload showed substantial improvement by 130% over the BLOB strategy (single workflow), and 160% improvement while running multiple workflows concurrently.

In our future work, we will extend our framework to address collaborative execution of workflows in the Cloud by multiple users, and efficient buffer of streaming data to reduce data transfers for batch reprocessing. We also plan to expand our experiments to multiple desktop machines to investigate further scale-out behavior.

Acknowledgements. We would like to thank Microsoft Research for providing resources on the Windows Azure Cloud platform to perform our experiments, and to Catharine van Ingen from MSR for her comments and feedback. We also thank Tim McPhillip for his help with the RestFlow workflow engine.

REFERENCES


According to personal communication between Bertram Ludäscher and Ewa Deelman, 7 pages are acceptable as a short paper.