Client + Cloud: Seamless Architectures for Visual Data Analytics in the Ocean Sciences

Keith Grochow[1], Bill Howe[1], Mark Stoermer[2], Roger Barga[3], Ed Lazowska[1]

[1] University of Washington, Computer Science Department,
Seattle, Washington, USA 98195
[2] Center for Environmental Visualization, Ocean Sciences Building,
1492 N.E. Boat Street, Seattle, Washington, USA 98195
[3] Microsoft Research, Microsoft Corporation
PO Box 91017, Redmond, WA, USA, 98073
{keithg, billhowe, lazowska}@cs.washington.edu

Abstract. Science is becoming data-intensive, requiring new software architectures that can exploit resources at all scales: local GPUs for interactive visualization, server-side multi-core machines with fast processors and large memories, and scalable, pay-as-you-go cloud resources. Architectures that seamlessly and flexibly exploit all three platforms are largely unexplored. Informed by a long-term collaboration with ocean scientists, we articulate a suite of representative visual data analytics workflows and use them to design and implement a multi-tier immersive visualization system. We then analyze a variety of candidate architectures spanning all three platforms, articulate their tradeoffs and requirements, and evaluate their performance. We conclude that although "pushing the computation to the data" is generally the optimal strategy, no one single architecture is optimal in all cases and client-side processing cannot be made obsolete by cloud computing. Rather, rich visual data analytics applications benefit from access to a variety of cross-scale, seamless "client + cloud" architectures.

Keywords: e-Science, Workflow, Scientific Visualization, Visual Data Analytics

1 Introduction

Science in every field is becoming data-intensive, motivating the use of a variety of data management, analysis, and visualization platforms and applications. There is no "one-size-fits-all" solution — the desired infrastructure requires cooperation between desktop GPUs for immersive, interactive visualization, server-side data processing, and massive-scale cloud computing. Applications that seamlessly span all three platforms, leveraging the benefits of each, are becoming the norm rather than the exception. Moreover, application components cannot be statically assigned to these resources — specific use cases motivate specific provisioning scenarios. For example, in "small data" conditions, local processing is ideal for simplicity, to reduce latency, to reduce load on shared resources, and — in the era of "pay-as-you-go" computing

— to reduce cost in real currency. However, as data is increasingly deposited in large shared resources, and as data sizes grow, reliance on local processing incurs significant transfer delays and may not be feasible at all. We advocate *seamlessness*, where data analysis pipelines transparently span a variety of platforms — client, server, cloud — and can adapt dynamically as the situation warrants.

Remarkably, there is little research on architecture, principles, and systems for seamless visual data analytics. Existing workflow systems can distribute data flow computations across a cluster [1-5], but consider local interactive visualization applications outside their scope. Existing visualization systems [6-8] lack data integration capabilities to access and manipulate data from different sources.

In this paper, we explore the design space for architectures spanning client, server, and cloud for visual data analytics in the ocean sciences. Our technology includes the Collaborative Ocean Visualization Environment (COVE) [9], the Trident Scientific Workflow Workbench running on both client and server, and the Microsoft Azure cloud computing platform. We compare various design choices, model their performance, and make recommendations for further research.

To inform the analysis, we recorded a benchmark of 16 visual data analytics scenarios gleaned from a multi-year collaboration with ocean scientists. From these scenarios we distilled a set of common sub-tasks and then implemented a selection of scenarios as visualization workflows in Trident and COVE using the subtasks.

For reasoning about performance, we model these visualization workflows as instances of a simple Fetch, Transform, Visualize pipeline common in the visualization domain, using this abstraction to derive a simple cost model based on data transfer costs and computation time. We use this cost model to design a set of experiments testing each workflow in a variety of multi-tier architecture scenarios using typical resources, then measure the time spent performing IO, data transformation, visualization, and data transfer.

We find that the role of the client remains critical in the era of cloud computing as a host for visualization, local caching, and local processing. The network bandwidth limitations found in practice currently dominate the cost of data analytics, motivating the need for pre-fetching and aggressive caching to maintain interactive performance necessary for immersive visualization applications. We also find that lack of a GPU is a severe limit to effective visual data analytics, suggesting that the generic hardware found in many cloud computing platforms are not a complete solution. Finally, we show that there is no "one size fits all" architecture that is satisfactory in all cases, motivating further research in dynamic provisioning and seamless computing.

Summary of contributions: We evaluate potential architectures for *seamless*, *multi-platform* visual analytics using a representative benchmark of workflows in the ocean sciences. We implemented these workflows in an integrated visualization and workflow system using COVE and Trident, and tested them on several candidate architectures involving client, server, and cloud resources

We make the following specific contributions:

- We present a test suite of representative visual data analytics tasks derived from a multi-year collaboration with ocean scientists;
- We describe a comprehensive visual data analytics system based on COVE, an immersive visualization environment, and Trident, a scientific workflow workbench, to support seamless multi-platform computing;

- We implement the test suite on the complete system across a variety of different architectures spanning client, server, and cloud;
- We experimentally compare these architectures using the test suite, report and analyze their performance, and conclude that seamless "Client + Cloud" architectures — as opposed to cloud-alone or client-alone — are an important consideration for visual data analytics applications.

2 Background and Related Work

Visualization. McCormick et al provide an early and influential call to arms for scientific visualization [10] in the face of large datasets. The authors' arguments are remain remarkably relevant today: They articulate a standard filter-map-render architecture, shown in Fig. 1, around which the majority of visualization systems today are designed [11].

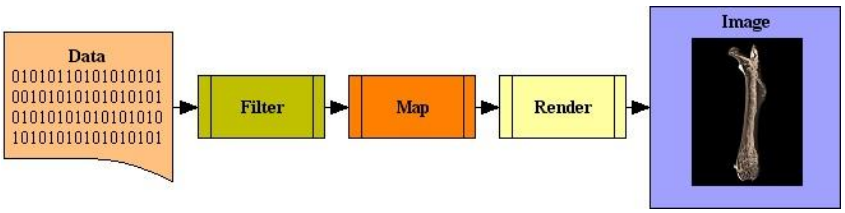

Fig. 1. The standard filter-map-render visualization dataflow pipeline model that turns scientific data into rendered images on the user's screen.

This architecture consists of three logical steps: *Filter* (data selection, extraction, and enrichment), *Map* (application of visualization algorithms to generate a viewable scene), and *Render* (generating a series of images from the logical scene). Today, the Map and Render steps are typically performed together using high-performance Graphics Processing Units (GPUs); we refer to these two steps together as simply "visualization." In the last two decades, although significant strides have been made in scientific visualization, the computational sciences are still severely limited by data visualization and post hoc analysis capabilities rather than the availability of cycles. Further, the HPC community no longer has a monopoly on "big data" problems. Advances in sensing technology in oceanography, biology, astrophysics, medicine — indeed, nearly every field of science and engineering — have contributed to an unprecedented situation: the availability of data is no longer the bottleneck to scientific discovery [12].

For visualization to "catch up," we need more flexibility in where and how visualization pipelines are executed — in parallel, on the client, on local servers, in the cloud. Visualization pipelines must incorporate more than just "Filtering"; they must perform arbitrary data processing, restructuring, manipulation, and querying — the capabilities associated with data management and workflow systems rather than pure visualization systems [3].

The requirements of visualization systems, analytics engines, and data retrieval systems are converging. The scientific visualization community is recognizing that

visualization systems must do more than just "throw datasets" through the rendering pipeline — that data restructuring, formatting, query, and analysis cannot be relegated to an offline "pre-processing" phase [13, 14]. Simultaneously, the data management community is recognizing the importance of incorporating visualization capabilities into data management systems, for two reasons. First, visualization is a critical method of interpreting large datasets, thanks to the acuity and bandwidth of the human visual cortex — humans cannot quickly "see" the patterns in a million tuples without a visual representation. Second, visualization pipelines typically reduce large datasets to relatively small images or sequences of images, so requiring the client to be solely responsible for visualization involves a significant data transfer cost.

Most research focuses on the largest-possible scale of visualization, leveraging powerful and specialized hardware, highly optimized but narrowly focused algorithms, and massively parallel distributed computing paradigms [14]. However, we observe that in practice, a significant amount of exploratory, ad hoc visual analysis is performed on commodity hardware connected to data stores by commodity network links. Software systems that can seamlessly bridge the gap between the desktop and larger-scale facilities are desperately needed.

Workflow. Workflow systems [1-5] provide a significant step forward in this regard, striving for several goals simultaneously. First, and perhaps most importantly, workflow systems attempt to raise the level of abstraction for scientist-programmers, allowing them to reason about their computational tasks visually as data flow graphs instead of syntactically as scripts. Second, workflow systems aim to provide reproducible research. Perhaps paradoxically, computational science tasks often proven to be less likely to be reproducible than laboratory protocols, due to diversity of languages, platforms, user skills, and usage scenarios. Expressed as a workflow (assuming agreement on the workflow system!), these protocols are easier to share, reuse, and compose than are raw scripts. Third, and most relevant to our discussion, workflow systems help to abstract away the execution environment, allowing workflow tasks to be executed on a variety of different platforms. For example, both the Kepler and Trident systems allow workflows to be submitted to a cluster for execution or be evaluated directly in the desktop environment.

However, these tools do not provide for interactive visualization on the client — workflows are modeled as batch jobs to be executed once. The VisTrails system [3], adopting the VTK visualization library [6] as a core plugin, has rich support for visualization. VisTrails also provides a powerful caching mechanism to support repeated execution and exploratory analysis. However, VisTrails, like other workflow systems, is essentially middleware. In this work, we operate at a broader scope, integrating a rich client GUI on the front end and consider various data sources (and therefore data transfer times) on the back end.

Workflow execution and optimization is a well-studied problem (c.f., [4, 15-17]), but these approaches typically ignore client-side processing and interactive visualization. We demonstrate that the local client remains an important resource. Further, optimization of workflow execution over heterogeneous environments is NP-complete [17]. We therefore adopt a simpler model and experimentally verify its accuracy.

Table 1: Use case scenarios for visual data analytics in oceanography

Scenario	Data	Analytics	Visualization
Data Archive Investigation	Low	High	High
Ecosystem Modeling	High	Medium	Medium
Ocean Observatory Simulation	High	Medium	Low
PCA for Instrument Placement	Low	High	Medium
Hydrographic Survey Analysis	Low	Medium	Medium
Observation and Model Comparison	High	Medium	High
Flow Field Analysis	Medium	Medium	High
Hydrographic Fluxes through a Volume	Medium	High	Medium
Seafloor Mapping	High	High	High

3 Ocean Science Requirements Analysis Methodology

The goal of our requirements analysis phase was to obtain a suite of real ocean data visualization and analysis scenarios, then translate them into workflows to measure the effectiveness of different visualization architectures. This effort involved many months of close collaboration with the scientists to not only capture answers to explicit questions, but also to observe them in their daily work. This observational study was necessary, we found, because crafting an effective scientific visualization is an activity difficult to capture procedurally. Many of the steps were not documented and were problematic for the scientists to recall at the time of the interview.

We worked with scientists at two different ocean science institutions: the Monterey Bay Aquarium Institute (MBARI) [18], and the University of Washington College of Ocean and Fisheries Sciences [19]. MBARI is the largest privately funded oceanographic organization in the world and acquires data through fixed and mobile instruments, ship based cruises, and occasional large-scale multi-institute projects. We worked with them on two such projects: The Autonomous Ocean Sampling Network (AOSN), which is a series of multi-month activities to measure the effectiveness of adaptive sampling in Monterey Bay, and in preparation for a multi-organization program in 2010 to study the interaction of typhoons with the ocean surface. At the University of Washington College of Ocean and Fisheries Sciences, we worked with one group building the NSF-funded Regional Ocean Observatory, and another group generating regional-scale simulations of the Puget Sound. Both of these organizations consist of primarily desktop system users who connect to a local network to share data when necessary. Both had also expressed interest in how cloud computing could help them currently and in the future.

We met with each of these groups on multiple occasions to collect examples of datasets, visualizations, and workflows. We also interviewed eleven members of the teams in depth to glean detailed requirements.

3.1 Identified Use Cases

The key result of this effort was a set of 16 data analysis scenarios spanning a wide range of requirements in oceanographic data visualization and analysis. Of these 16, we focused on 9 that had relatively similar workflow needs based on our analysis and discussions with the scientists. These scenarios were broadly measured with regards to how data-intensive, computation-intensive, and visualization-intensive they were which is summarized in Table 1. Data-intensive, in this case, is still "desktop-scale" — in the order of hundreds of megabytes per operation rather than tens of terabytes.

Table 2: Workflows tested

Workflow	Description
Advect Particles	Display the path of several particle paths based on a simulated or observed flow model.
Combine CODAR	Merge and filter observed flow patterns on the ocean surface and visualize surface current changes.
Combine Models	Combine overlapping model files to create a time series and view selected iso-surfaces in the data.
Compare Models	Compare two models simulated on different grids and view the differences between the models.
Compare Data to Model	Resample model data based on observed data locations and compare the observed data to the simulation.
Filter Model	Run a set of filters on a data file to clean and smooth the data.
PCA	Reduce several thousand dimension model to 5 dimensions using PCA and view the resulting differences.
Regrid Model	Regrid arbitrarily gridded model to a rectangular lat/lon grid.
Subsample Terrain	Resample and filter terrain data and then visualize it on a globe project.
Supersample Data	Interpolate data in a model using splines to create a higher resolution version of the model
Verify Model	Run a combination of filters and tests on new simulation output to determine its fidelity
Vertical Section	Extrapolate to create a vertical section from a vessel or glider data stream and visualize in the collection location

As can be seen from the table, there is no consistent theme to the more intensive areas of the workflows even though they shared very similar underlying tasks. This is due in some part to the nature of oceanographic data. There are simulations of the ocean which are very data-intensive, involving multiple terabytes of simulation output. Observed data is significantly smaller since it is very expensive to obtain and usually quite sparse, requiring aggressive extrapolation and interpolation to compare with higher-coverage datasets such as simulation output. We also find that analytics are not usually compute-bound unless one is performing high-dimensional matrix-oriented computations such as principal component analysis (PCA). Visualization needs vary from simple 2D plots up to animations of multiple 3D iso-surfaces.

From this suite of scenarios, we derived 43 re-usable components (called *activities*) and used them to recreate the scenarios. These activities can be linked together interactively to carry out filtering of the raw data to create visualization ready data products. Some of the tasks supported were subsampling, supersampling, cropping,

filtering, masking, scaling, resampling data to match other data sample points, and file conversion from their standard formats. Some of these activities are non-compute intensive while others, such as resampling of simulations, can be quite compute-intensive based on the irregular arbitrary grids that simulations often have. We also provide more specific activities such as PCA and creation of vertical cross sections by supersampling instrument collected data samples. This activity set is now part of the standard oceanographic library that is released as part of Microsoft's Trident Workflow package [2].

From these activities, we chose a set of 12 visualization-based workflows that provide a cross-section of the types of workflows we observed or collected from the scientists. These workflows each consisted of from 8 to 20 activities and comprised the use cases for our visualization architecture described in the next section. Each of the workflows is listed below along with a brief description.

3.3 Analysis

Each of the workflows we derived from interaction with the scientists can be considered an instance of the very simple model of a visualization pipeline shown in Fig. 1. Each instance consists of three components: a data source (Data), a workflow (Filter), and a visualization (Map and Render). This model is minimal in the sense that the data source and visualization steps cannot generally be folded into the context of the workflow engine. Each data source, in general, may be a remote service outside the scope of the workflow system, and the visualization step, as in this case, may be an interactive visualization environment. Neither of these components is adequately modeled by the dataflow graph model of computation adopted by workflow tools.

Informed by the basic Data, Filter, Map, Render visualization pipeline, we model visual analysis tasks in terms of four logical software components: Data Store, Workflow, Visualization, and Client arranged linearly according to dataflow. We map these components onto a physical architecture consisting of three resources: Cloud, Server, and Client.

An *architecture configuration*, or simply *configuration*, is a mapping from components {Data Store, Workflow, Visualization, Client} to {Local, Server, Cloud}. The cost with respect to a task is the sum of the time to execute the workflow, the time to execute the visualization, and the time to transfer the data between each step. That is:

```
COST = RAWSIZE / BANDWIDTH_DATA_WF
       + WF_WORK(RAWSIZE) / PROCESSOR_WF
       + (WF_RATIO * RAWSIZE) / BANDWIDTH_WF_VIZ
       + (IMAGECNT * IMAGESIZE) / BANDWIDTH_VIZ_CLIENT (1)
```

RAWSIZE is the size in bytes of the input dataset. BANDWIDTH_DATA_WF and BANDWIDTH_WF_VIZ are the bandwidth between the data source/workflow system and the workflow system/visualization system. WF_RATIO is the ratio of the size of the input to the size of the output of the filter step; it may be greater than 1. PROCESSOR_WF and PROCESSOR_VIZ are the processor speeds for the respective machines, accounting for the potentially significant difference between

server machines and client machines. WF_WORK and VIZ_WORK are functions of data size and return the (approximate) number of instructions required to process their input. These functions can be estimated precisely through curve fitting, sampling, or provided by the user directly [20]. These functions are typically polynomial in the size of the input data, but we find that even rough linear estimates of the workflows provide a reasonable estimate.

Although this model precisely describes the cost of the workflow, a simpler model based *only* on data transfer overhead can capture the relative cost between different architectures. In this case, the model is

```
COST = RAWSIZE / BANDWIDTH_DATA_WF
       + (WF_RATIO * RAWSIZE) / BANDWIDTH_WF_VIZ
       + (IMAGECNT * IMAGESIZE) / BANDWIDTH_VIZ_CLIENT (2)
```

In Section 5, we will show experiments that justify this simplification for certain configurations.

4 System Design

To implement a system to measure the cost components or workflow set, we leverage three existing systems: the COVE visualization system, the Microsoft Trident workflow system, and Microsoft Azure Cloud Service. These are each described in more detail below.

4.1 Visualization with COVE

The Collaborative Ocean Visualization Environment (COVE), shown in Fig. 2, is a system designed in close collaboration with scientists to provide support for oceanographic data visualization and planning. For ease of the user, the interface is based on the Geo-browser interface applied successfully for data visualization in applications such as Google Earth and Microsoft's Virtual Earth. These systems provide an intuitive, zooming, multi-scale interface for geo-positioned data. COVE provides all the essential features of these geo-browser systems, as well as enhancements informed by collaboration with ocean scientists.

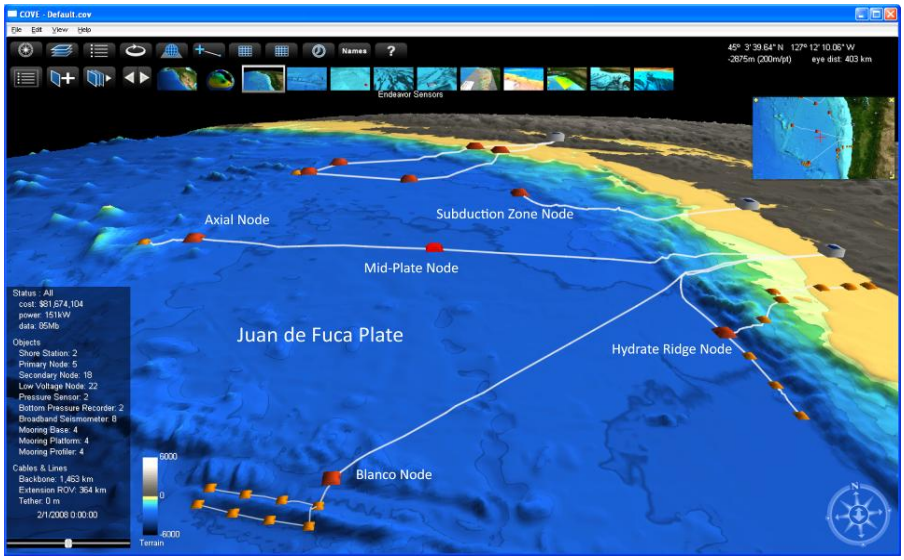

Figure 2: COVE is a rich cross platform client that provides the ability to create geo-positioned visualizations of scientific data, seafloor terrain, images, and instrument layout

In particular, COVE incorporates better support for the time and depth dimensions. Visualizations can be animated and synchronized in time. COVE also provides extensive terrain visualization support, as the bathymetry of the ocean floor is constantly changing and each scientist may prefer a different source of terrain information. To provide more visual cues for the underwater terrain there are depth-based color gradients, contour lines, or binned color values, as well as user adjustable shading and terrain detail to enhance visualization of the seafloor.

To enable experiment planning, asset deployment and tracking, and observatory design, enhanced interactive layout facilities are provided. To support track and cable layout, COVE provides a large selection of smart cable and track types. These conform to the terrain, and positioning handles are available for maneuvering the cable around obstacles such as trenches. To provide instant feedback (e.g., budget and current cost), heads-up displays are provided during editing sessions.

To help share visualizations throughout the team, anything created in COVE can be uploaded to a server to be viewed by other members of the team. Other users can then download the visualization script and datasets to jumpstart derivation of new visualizations for their own needs. To keep apace of changes on the server, scientists can register to be alerted when changes to data or locations of interest are committed.

COVE has been successfully deployed for a variety of tasks. It was a key tool in the design of an NSF-funded deep-water ocean off the northwest coast of the United States, and has also been a part of two ocean expeditions. The first expedition mapped sites for the deep-water observatory and the second explored ways to support deep ocean navigation while exploring volcanic sites on the Juan de Fuca plate.

While quite successful in these limited deployments, limitations became apparent. While a simplified interface was empowering to novices, expert users required some

form of extensibility. Also, as datasets began to grow in size, scalability problems associated with a desktop-only deployment of COVE emerged. To meet these needs, we integrated the Trident workflow system for data analysis and pre-processing.

4.2 Workflow with Trident

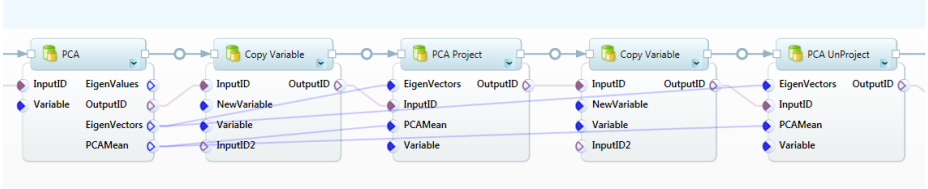

Fig. 3. This image displays an example of the interactive workflow editing interface of Trident.

Workflow systems have been developed to aid scientists in managing their computational activities. Scientific workflow offers a number of advantages over traditional script-based approaches, including visual programming, improved reusability, provenance, and web service orchestration, and execution in heterogeneous environments.

The Trident Workflow system, developed at Microsoft Research, is a domain-independent workbench for scientific workflow using Microsoft's Windows Workflow Foundation. In addition to the features common to many workflow systems, it also provides automated provenance capture, "smart" re-execution of different versions of workflow instances, on-the-fly updateable parameters, monitoring of long running tasks, and support for fault-tolerance and failure recovery.

Trident can be executed on the local desktop, on a server, or on a High Performance Computing (HPC) cluster. It currently runs on the Windows OS using the .NET API, with SQL Server for data storage and provenance capture. Interactive editing and management of workflows is available through a set of programs that are part of the Trident suite. Trident provides cross-platform support using Silverlight, a freely-downloadable cross-browser, cross-platform, and cross-device plug-in for delivering .NET-based applications over the Web.

"Headless" cross platform support is also available through a web service interface developed as part of this effort. This interface allows execution and job control through a RESTful API. For example, a user can login, select a desired workflow, monitor its progress, poll for created data products, and retrieve data products for local use using GET and POST calls. This is the interface used by COVE to connect to Trident in order to provide cross-architecture access to Trident.

4.3 Cloud Services with Azure

Azure is a cloud computing platform offering by Microsoft. In contrast to Amazon's suite of "Infrastructure as a Service" offerings (c.f., EC2, S3), Azure is a "Platform as a Service" that provides developers with on-demand compute and storage to host, scale, and manage web applications on the internet through Microsoft datacenters. A

primary goal of Windows Azure is to be a platform on which ISVs can implement Software as a Service (SaaS) applications. Amazon's EC2, in contrast, provides a host for virtual machines, but the user is responsible for outfitting the virtual machine with the software needed for their task.

Windows Azure has three parts: a Compute service that runs applications, a Storage service, and a Fabric that supports the Compute and Storage services.

To use the Compute service, a developer creates a Windows application consisting of *Web Roles* and *Worker Roles* using, say, C# and.NET or C++ and the Win32 APIs. A Web Role package responds to user requests and may include an ASP.NET web application. A Worker Role, perhaps initiated by a Web Role, runs in the Azure Application Fabric to implement parallel computations. Unlike other parallel programming frameworks such as MapReduce or Dryad, Worker Roles are not constrained in how they communicate with other workers.

For persistent storage, Windows Azure provides three storage options: *Tables*, *Blobs*, and *Queues*, all accessed via a RESTful HTTP interface. A table is akin to a scalable key-value store, a Blob is a file-like object that can be retrieved, in its entirety, by name, and a Queue simplifies asynchronous inter-communication between workers. The Windows Azure platform also includes SQL Azure Database (previously known as SQL Data Services, offering standard relational storage based on SQL Server. Since we are not evaluating Azure in this work, we model data sources as Blobs and simply retrieve them for processing by our workflow engine.

4.4 Physical Architecture Variants

With these systems there are a variety of configurations that can be created to run the visualization workflows enumerated in the previous section. In Figure 4, we illustrate the six architectures we evaluate.

In the first configuration, all the data and visualization is being handled locally. This is the most common visualization mode we noted with the scientists. It avoids network latency or cross platform communication issues. The disadvantage is that computation and data size are limited by available cores and local storage capacity, respectively.

In the second the data has been moved remotely to either the web or cloud. This allows data larger data sizes and also allows sharing of the data with other researchers, but with the overhead of downloading the data to the workflow service.

In the third the computation has been moved remotely to either the web or cloud and co-located with the data. This leverages the computational and storage capabilities of the remote platform and removes the overhead of moving raw data. There is the cost of downloading the resultant filtered data.

In the fourth the computation has been moved remotely to a server and the data stored on in cloud storage. One could also reverse these with Trident in the cloud and the data on a server. This allows the most flexibility to optimize the choice of platform based on cost and needs. It also is the most sensitive to network speeds since raw and filtered data are both transferred.

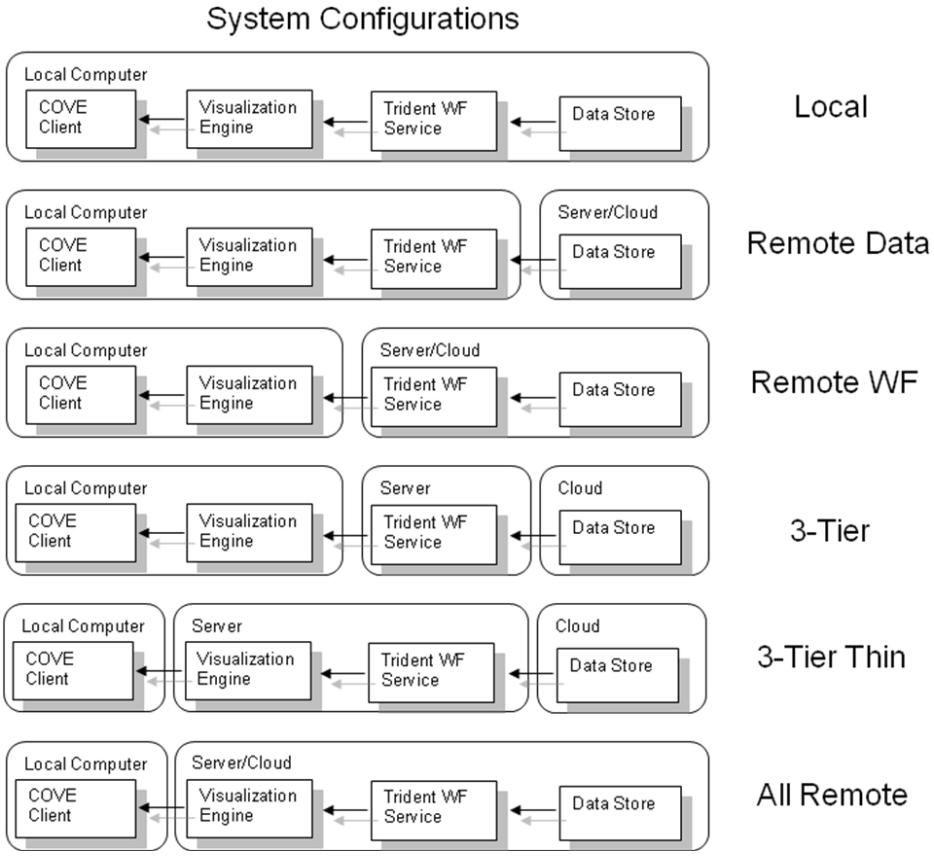

Fig. 4. The 6 evaluated configurations of the COVE+Trident+Azure system.

3-Tier Thin provides the ability to move the data and visualization handling to a server, possibly with fast graphics capability, and place the data on cloud storage. This configuration is useful for a thin client environment such as a browser or phone interface, but requires a fast connection between the cloud and the server.

The final configuration allows all the data to be handled in a cloud computing or server environment with a minimum of network overhead since only the visual product is returned. The drawback is that the environment may not provide graphics support for fast visualization creation. Our experiments with this configuration were carried out on server without graphics support.

4.5 Data Model

Our data model is essentially file-oriented. On Azure, each file is stored as a Blob. On other platforms, each file is stored on local disk and referenced with standard filename conventions. In either case, we access files using HTTP.

Files are downloaded by the workflow system from the data store and cached locally. The workflow system then accesses them from the local cache. This could be optimized to reduce the overhead of reading and writing from the disk, but the local storage also allows for re-use of cached files in future workflows. Futher, we observe in Section 5 that the IO overhead is small relative to the overall cost of the workflow.

Similarly, the resulting data products are cached locally by the workflow service and made available for download using a RESTful API. Although Trident provides access to SQL server for data storage, we found the current implementation for serializing and deserializing large files to the database to be prohibitively slow. Instead, we implemented a multi-threaded file-based data storage solution that significantly improved IO performance. All experiments were conducted using the file-based storage solution.

Trident by default loads all data into memory to allow pointer-based access. Large files can exceed physical memory and lead to thrashing. For our trident workflow activities, we instead use a lazy loading strategy. We load just a descriptive header when opening a file, and read in the sections of the file only when necessary to carry out computation. This technique reduces memory needs and prevents thrashing.

The data files we access are taken directly from our collaborators. Each dataset is represented as a NetCDF [21] file or in simple binary and textual table data formats. NetCDF files are a very common format in the ocean sciences and allow us to use publicly available libraries for data access. We also use the CF Metadata naming conventions or convert to them on load to make identifying position and time variables easier.

4.6 Programming Model

The core activity methods are in a windows dynamic link library for increased performance. The object interface for from libraries is exported through the Simplified Wrapper and Interface Generator (SWIG) to a set of .NET managed C# activities. Each activity typically accesses a single method in the library.

Each activity has a set of inputs and outputs that can be declared explicitly by the user or implicitly through composition with another activity (e.g., the output of the file load activity may connect to the input of the resample activity.) The activities may be linked together interactively using the Trident Workflow Composer or by editing the XML based workflow description. The activities are executed either serially or in parallel according to the instructions in the workflow specification.

Our activity model design also made it very easy to expose the same functionality available in the workflow system to a scripting language. For example, we export a python interface with SWIG rather than a C# interface and then embed a python interpreter in COVE that can run scripts that call Trident methods. This technique has been useful for users that do not use Windows.

5 Experimental Analysis

We tested all 12 of our benchmark workflows using the six architecture configurations in Fig. 4. The overall performance results are shown in Figure 7. This figure displays the average time for the workflows on each in the suite on broken into the 5 major cost factors. Each workflow is associated with four bars, one for each of the following architectures: Local Data, Web Data, Web WF, and 3-Tier.

Setup. We instrumented COVE and all of the activities we added to Trident to measure wall clock time. The instrumentation overhead was negligible. We measured the time for network transmission (NET), system I/O (IO), workflow calculation (CPU), and visualization creation time (GPU). We used the three systems for the tests listed in Table 3 as the Local Machine, Web Server, and Azure Web Role in our architecture configurations.

Table 3: Physical Specification of Machines

Machine	Description
Local Machines	Apple Macbook Pro Laptop running Windows 7 (32 bit) Intel Core Duo T2600 CPU 2.16 GHz, 2GB RAM, Radeon X1600, 256 Mb memory Internet Connection: 11.43 Mb/Sec in, 5.78 Mb/Sec out
Web Server	HP PC running Windows Server 2008 R2 Enterprise Intel Core Duo E6850 CPU @ 3.00 GHz, 4 GB RAM NVIDIA GeForce 9600 GT, 512 Mb memory Internet Connection: 94.58 Mb/Sec in, 3.89 Mb/Sec out
Azure Web Role	Intel PC running Windows Server 2008 R2 Enterprise Intel 1.5-1.7 GHz, 1.7 GB RAM, No Video System Internet Connection: .85 Mb/Sec in, 1-2 Mb/Sec out

Data sizes: The data sizes used for each workflow appear in Fig. 6, averaging around 150MB per task. Typical datasets to include a single time step of an ocean simulation, a set of "glider tracks" from an Autonomous Underwater Vehicle (AUV), or a terrain model for region. All datasets pertain to the Pacific Northwest region and are "live" in the sense that they are actively used by scientists.

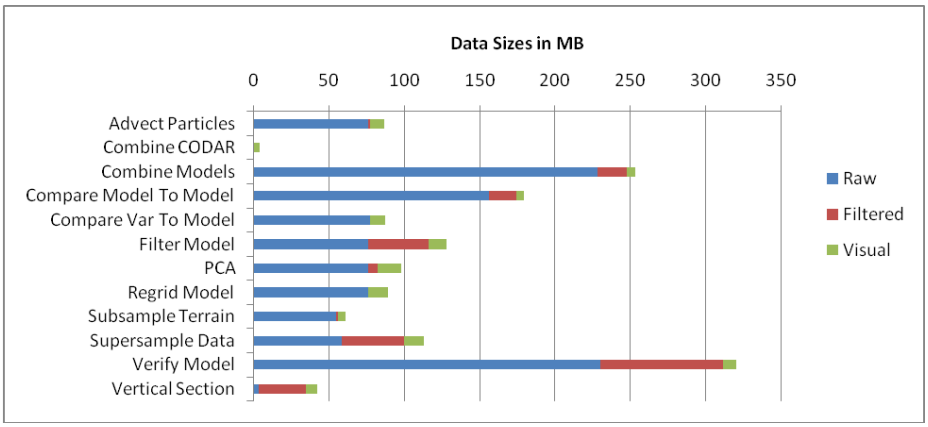

Fig. 5. Data sizes used in the experiments. Each bar is broken into three sections: the Raw data size, the filtered data size generated by the workflow, and the size of the final result generated by the visualization.

Summary We answer the following questions: (1) Is one architecture preferable for all of our visual analytics benchmarks? (2) What role does client-side processing have in cloud- and server-oriented analytics? (3) Does access to a GPU strongly affect performance for visual analytics workflows? (4) Does the simple cost model derived in Section 3 accurately represent performance? Our results show that: (1) There is no "one size fits all" architecture — the appropriate configuration depends on workflow characteristics (Figure XX); (2) client-side processing is a crucial resource for performance, assuming data can be pre-staged locally to minimize transfer costs (Figure YY); (3) access to a GPU strongly affects the performance of visual data analytics workflows, meaning that generic, virtualized cloud-based resources are not ideal (Figure ZZ); (4) the simple cost model is sufficient to capture the behavior of these workflows, and that the cost is generally dominated by data transfer times.

5.1 There is No "One Size Fits All" Architecture

The diversity of workflows in the benchmark illustrate that multiple architecture configurations must be supported in practice. Although local processing outperforms other configurations due to data transfer overhead, this configuration is not always viable. Among the alternatives, no one configuration is best in all cases. In the Vertical Section workflow, for example, the output o the filter step is larger than its input, motivating an architecture that pulls data down from remote locations before processing, contradicting the conventional wisdom that one should "push the computation to the data." In terms of the cost model, this distinction is captured by the WF_RATIO parameter. In Figure 7, the time profile for two workflows are displayed: one with WF_RATIO < 1, and the other with WF_RATIO > 1. For WF_RATIO < 1, the preferred (non-local) configuration to minimize transfer overhead is remote wf, where the data is processed on the same machine where it resides. However, when WF_RATIO > 1,the preferred configuration is to remote data, where data is sent to the client directly for processing.

The 3-tier configuration in these examples appears to be universally bad, but asymmetric processing capabilities between server and client can make up the difference. For example, the PCA workflow is highly compute bound, and therefore benefits from server-side processing at the middle tier.

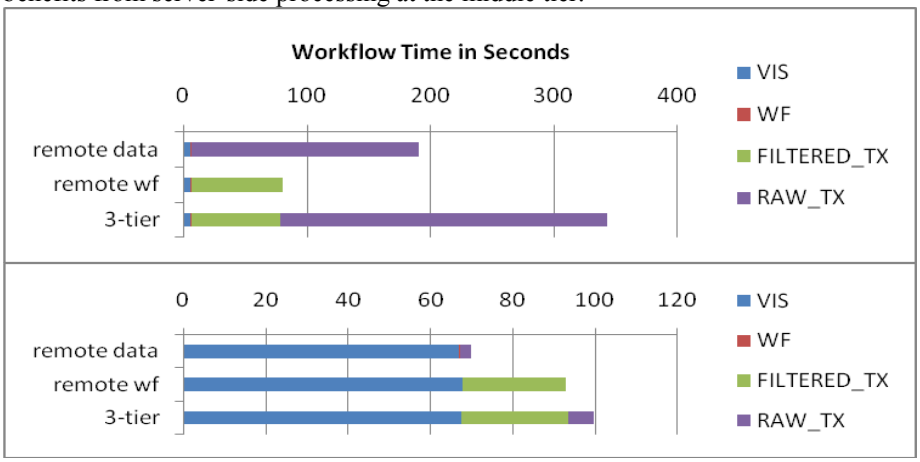

Fig. 6. Time profile comparison of a workflow with WF_RATIO < 1 on the top and WF_RATIO > 1 on the bottom. When WF_RATIO < 1, the preferred (non-local) strategy is to push the computation to the data using the remote wf configuration. When WF_RATIO > 1, the better strategy is to bring the data to the computation using the remote data configuration.

5.2 Client-side Processing Improves Efficiency

At these modest data sizes, the local data configuration performed well in all cases. Figure 8 shows the average performance across all benchmark workflows. However, local processing is only appropriate for small datasets that are either private or have been pre-staged on the user's machine. Since the trend in ocean sciences (and, indeed, in all scientific fields) is toward establishing large shared data repositories, we believe that aggressive pre-fetching and caching on user's local machines will become increasingly important. As part of our ongoing research, we are data scales, and for datasets that that are available locally, either by are always available locally or will soon be impossible for all but the smallest, most personal datasets. The data needed by one's current task will increasingly be found on large shared repositories, and we must identify the appropriate architecture configuration to use in this case.

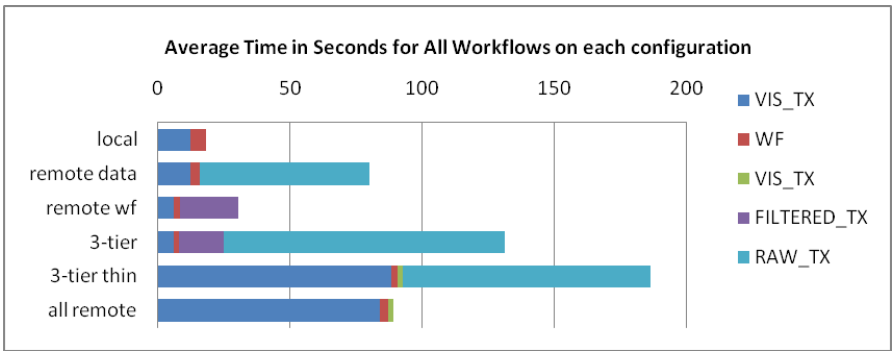

Fig. 7. The average runtime of all 12 workflows for each of the 6 architecture configurations is dominated by data transfer overhead. The local configuration, although not necessarily feasible in practice, eliminates this overhead and therefore offers the best performance. This result suggests that aggressive caching and pre-fetching methods should be employed to make best use of local resources.

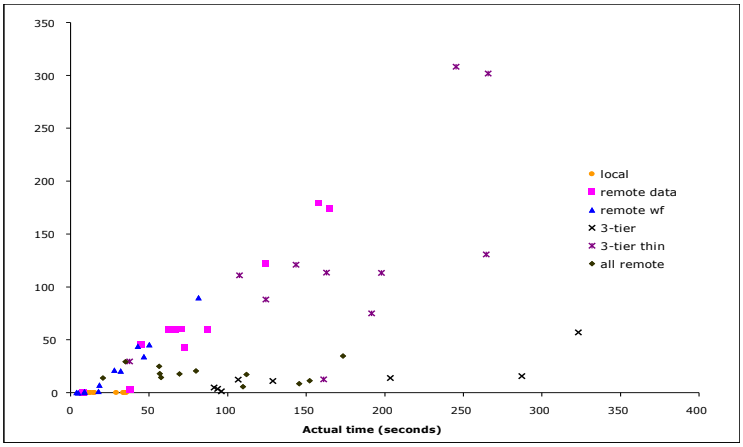

Fig. 9. Scatter plot of estimated results from equation versus the actual results.

5.2 Visual Analytics Benefit from GPU-Based Processing

Workflows involving significant visualization processing benefit from having access to GPUs. Although GPUs are becoming popular for general computation due to the their vector processing capabilities, they can be used for visualization tasks without customization. Using the instrumented COVE and Trident platform, we tested whether having access to a GPU would improve performance for the visual data analytics tasks. In particular, we allowed COVE and Trident to run "headless" as command line programs and performed rendering in software using the Mesa 3D library, the state-of-the-art 3D library. On average, the workflows ran 5x faster overall with access to a GPU. The visualization portion of the work ran 9x faster. This result suggests that the generic environment typically found on cloud computing platforms may be insufficient for visual data.

5.2 A Simple Cost Model Informs Architecture Decisions

The cost model presented in Section 3 is very simple, capturing each workflow as just a two-step process: filtering, and visualization. We allow the source data and each of these two steps to be located on any of the three tiers in the client-server-cloud pipeline, subject to technical constraints. Despite its simplicity, we find that this cost model adequately describes the computations, suggesting that an "architecture optimizer" could be based on it.

In Figure 9, we plot the estimated running time against the actual measured times using a model that ignores everything except for transfer times. The relationship is essentially linear, underestimating for CPU and Viz-heavy workflows, as well as fully local configurations that do not require any data transfer.

6 Conclusions and Future Work

Overall, we conclude that cloud-based platforms must be augmented with significant local processing capabilities for maximum performance. Due to the overhead of data transfer, access to GPUs for high-performance visualization, and the interactive nature of interactive visual data analytics, "Client + Cloud" architectures are appropriate for maximizing resource utilization and improving performance.

We base our conclusions on 1) a comprehensive, multi-year collaboration with ocean scientists from which we gleaned a suite of representative workflows, 2) a complete visual ocean analytics system involving immersive visualization capabilities in COVE and a flexible workflow environment in Trident, and 3) a set experiments testing each workflow in a variety of client, server, and cloud configurations.

Based on these results, we are pursuing an architecture optimization framework that will dynamically distribute computation across the client-server-cloud pipeline to maximize utilization and improve performance. We distinguish this work from the heterogeneous resource scheduling problem by focusing on interactive visualization and a domain-specific and realistic suite of workflows.

Acknowledgments. The authors wish to thank Microsoft's Technical Computing Initiative, University of Washington School of Oceanography, and Monterey Bay Aquarium Research Institute for data and technical advice. This work was supported in part by grants from Microsoft Corporation, the National Science Foundation (CluE grant and OOI grant), and a subcontract from the Ocean Observatories Initiative through UCSD and Woods Hole Oceanographic Initiative (OOI Grant #).

References

[1] *The Triana Project*. Available from: http://www.trianacode.org.

[2] Barga, R.S., et al., *Trident: Scientific Workflow Workbench for Oceanography*, in *IEEE Congress on Services - Part I*. 2008, IEEE Computer Society: Los Alamitos, CA, USA. p. 465-466.

[3] Bavoil, L., et al., *VisTrails: Enabling Interactive Multiple-View Visualizations*, in *IEEE Symposium on Visualization*. 2005.
[4] Deelman, E., et al., *Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems.* Scientific Programming Journal, 2005. **13**(3): p. 219-237.
[5] Ludascher, B., et al., *Scientific Workflow Management and the Kepler System.* Concurrency and Computation: Practice & Experience, 2005.
[6] *The Visualization Toolkit*. Available from: http://www.vtk.org.
[7] *ParaView*. Available from: http://www.paraview.org/.
[8] *MayaVi*. Available from: http://mayavi.sourceforge.net/.
[9] Grochow, K., et al., *COVE: A Visual Environment for ocean observatory design.* Physics Conference Series, 2008. **125**: p. 012092-012098.
[10] McCormick, B.H., T.A. DeFanti, and M.D. Brown, *Visualization in Scientific Computing.* Computer Graphics, 1987. **21**(6).
[11] Abram, G. and L. Treinish, *An Extended Data-Flow Architecture for Data Analysis and Visualization* in *IEEE Symposium on Visualization*. 1995.
[12] Gray, J., et al., *Scientific Data Management in the Coming Decade*. 2005, Microsoft MSR-TR-2005-10.
[13] Bethel, E.W., et al., *High Performance Visualization Using Query-Driven Visualization and Analytics*. 2006, Lawrence Berkeley National Laboratory: Berkeley, CA, USA, 94720.
[14] Ma, K.-L., et al. (2009) *Next-Generation Visualization Technologies: Enabling Discoveries at Extreme Scale* SCIDAC Review.
[15] Bright, L. and D. Maier, *Efficient scheduling and execution of scientific workflow tasks*, in *SSDBM'2005: Proceedings of the 17th international conference on Scientific and statistical database management*. 2005, Lawrence Berkeley Laboratory: Santa Barbara, CA. p. 65-74.
[16] Langguth, C., P. Ranaldi, and H. Schuldt, *Towards Quality of Service in Scientific Workflows by Using Advance Resource Reservations.* Services, IEEE Congress on, 2009. **0**: p. 251-258.
[17] Wu, Q., et al., *Optimizing Distributed Execution of WS-BPEL Processes in Heterogeneous Computing Environments*, in *QSHINE*. 2009. p. 770-784.
[18] *Monterey Bay Aquarium Research Institute*. Available from: http://www.mbari.org.
[19] *College of Ocean and Fishery Sciences, University of Washington*. Available from: http://www.cofs.washington.edu/.
[20] Jihad, B. and O. Kinji, *Cost estimation of user-defined methods in object-relational database systems.* SIGMOD Record, 1999. **28**(3): p. 22–28.
[21] Jenter, H.L. and R.P. Signell. *NetCDF: A Public-Domain-Software Solution to Data-Access Problems for Numerical Modelers*. 1992.