Client + Cloud: Evaluating 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; {keithg, billhowe, lazowska}@cs.washington.edu
[2] Center for Environmental Visualization, Ocean Sciences Building,
1492 N.E. Boat Street, Seattle, Washington, USA 98195; mstorm@u.washington.edu
[3] Microsoft Research, Microsoft Corporation
PO Box 91017, Redmond, WA, USA, 98073; barga@microsoft.com

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.

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. This is particularly true for the Earth sciences that involve a host of observational and numeric modeling systems. A comprehensive infrastructure addressing this need requires cooperation between desktop Graphics Processing Units (GPUs) for immersive, interactive visualization, server-side data processing, and massive-scale cloud computing. The requirement that applications seamlessly span all three platforms, leveraging the benefits of each, is 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. We advocate *seamlessness*, where data analysis pipelines transparently span a variety of platforms — client, server, cloud — and can be reconfigured dynamically as the situation warrants — either manually as with our current system or automatically based on estimated cost.

Remarkably, there is little research on architecture, principles, and systems for seamless visual data analytics. Existing workflow systems are dataflow-oriented [1-5]; they do not subsume client-side interactive visualization applications such as Google Earth. 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 [1] running on both client and server, and the Microsoft Azure cloud computing platform [10]. We compare various design choices, model their performance, and make recommendations for further research.

To inform the analysis, we defined 9 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 the scenarios as representative visualization workflows in Trident and COVE.

We model these visual data analytics workflows as instances of a Data-Workflow-Visualization-Client pipeline, and use this abstraction to derive a simple cost model based on data transfer costs and computation time. We then use this cost model to design a set of experiments testing each workflow in a variety of multi-tier architecture scenarios using typical resources, measuring both computation and data transfer costs at each step.

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 frequently 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 confirm that a GPU is crucial for efficient visual data analytics, suggesting that the generic hardware configurations 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, Trident, a scientific workflow workbench

to support seamless multi-platform computing, and Microsoft Azure, a cloud computing platform that we use for serving data and limited computation;

- 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 the importance of visualization [11] in the face of large scientific datasets. Stemming from this need, computer visualization researchers articulated a standard data visualization pipeline architecture around which the majority of visualization systems today are designed [12]. This architecture consists of three logical steps: *Filter* (selection, extraction, and enrichment of data), *Map* (production of a spatial representation of the data using visualization algorithms), and *Render* (generation of a series of images from the spatial representation). Today, the Map and Render steps are typically performed together using high-performance GPUs; we refer to these two steps together as simply "visualization."

Visual Data Analytics pipelines, in contrast to pure 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 as opposed to pure visualization systems [2].

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 data points without a visual representation. Second, since visualization pipelines typically reduce large datasets to relatively small images or sequences of images, requiring the client to be solely responsible for visualization creates a significant data transfer cost.

Some proposed systems provide optimization and control of distributed visualization pipelines [15, 16], but are restricted to specialized visualization algorithms rather than a general purpose framework, as our collaboration with ocean scientists mandates.

Workflow. Workflow systems [1-5] provide a significant step forward in this regard, striving for several goals simultaneously. First, 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 tasks are often more difficult to reproduce 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, Kepler and Trident systems allow workflows to be submitted to a cluster for execution or be evaluated directly in the desktop environment [1, 4].

However, these tools do not provide for interactive visualization on the client — workflows are typically executed as batch jobs. More recently, the VisTrails system [2], adopting the VTK visualization library [8] as a core plugin, has added richer support for visualization. VisTrails also provides a powerful caching mechanism to support repeated execution and exploratory analysis. However, VisTrails does not consider visual analysis pipelines that span multiple platforms in one execution.

Workflow execution and optimization is a well-studied problem [3, 17-19], but these approaches typically ignore client-side processing and interactive visualization. We demonstrate that the local client remains an important resource. Further, since optimization of workflow execution over heterogeneous environments is NP-complete [19], we adopt a simpler model and experimentally verify its accuracy.

3 Ocean Science Requirements

The goal of our requirements analysis was to obtain a representative suite of real ocean data visualization and analysis scenarios, then measure the effectiveness of different visualization architectures on their performance. This was part of a multi-year study to determine requirements for more effective science visualization tools. We met with groups of scientists 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. This close collaboration was instrumental to our success, as many of the workflows were not documented and were often problematic for the scientists to recall from memory.

Our work took place at two different ocean science institutions: the Monterey Bay Aquarium Institute (MBARI) [20], and the University of Washington College of Ocean and Fisheries Sciences [21]. 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 MBARI on two such projects: The Autonomous Ocean Sampling Network (AOSN), which is a program to design and build an adaptive, coupled observation/modeling system. This program involved a series of multi-month activities to measure the effectiveness of adaptive sampling in Monterey Bay. A second MBARI effort involves the 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 Regional Scale Nodes (RSN) portion of the NSF-funded Ocean Observatories Initiative, and another group generating regional-scale simulations of the Puget Sound. Both of these institutions consist of primarily desktop system users who connect to a local network to share data when necessary. Both also expressed interest in how cloud computing could help them on current and future projects.

3.1 Abstract Use Scenarios

A key result of this interaction with ocean scientists was a set of 16 data analysis scenarios spanning a wide range of requirements in oceanographic data visualization and analysis. To make our investigation more tractable, we focused on 9 scenarios that had relatively similar workflow needs based on our analysis and discussions with the scientists. These scenarios are listed in Table 1 along with a short description.

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

Scenario	Description
1) *Data Archive Analysis*	Analyze existing collections of observed and simulated data
2) *Ocean Modeling*	Generate more accurate and denser ocean simulations
3) *Observatory Simulation*	Simulate ocean observatory data collection from existing data
4) *PCA Sensor Placement*	Determine optimal sensor placement using PCA modeling
5) *Hydrographic Analysis*	Estimate larger ocean effects based on limited observed data
6) *Data Comparison*	Compare observed and simulated data sets for integrity
7) *Flow Field Analysis*	Measure changes over time based on ocean currents
8) *Hydrographic Fluxes*	Measure changes over time in a specific ocean volume
9) *Seafloor Mapping*	Generate detailed terrain maps from collected sensor points

What we observe in our study is that even though the scenarios shared very similar underlying tasks, they are difficult to categorize as data-intensive, computation-intensive, or visualization-intensive. This difficulty is due in some part to the heterogeneous nature of oceanographic data. Simulations of the ocean are very data-intensive, producing multiple terabytes of simulated output. Most observed data, in contrast, is significantly smaller since it is expensive to obtain and usually sparse, requiring aggressive extrapolation and interpolation to determine ocean effects. Therefore data sizes in a scenario often show extreme variability from task to task. We also find that while large datasets increase computation time as expected, analytics are not usually inherently compute-bound in these scenarios. Visualization needs vary from simple 2D plots up to animations of geographically located datasets with multiple 3D iso-surfaces of ocean parameters. The choice of visualization primarily depends more on a current research need rather than the specific scenario category, making it difficult to build specialized applications.

3.2 Concrete Workflows

From this suite of scenarios, we derive 43 re-usable components (called *activities*) in order to recreate the scenario visualizations. These activities are linked together to carry out filtering of the raw datasets to create visualization ready data products. Some of the activities supported were subsampling, supersampling, cropping, filtering, masking, scaling, merging, and resampling data to match other data sample points. Some of these activities are not compute-intensive while others, such as resampling of simulations, can be quite compute-intensive due to the use of irregular grids in ocean simulation and the size of the simulated datasets. We also provide more ocean-science-specific activities such as *particle advection* to map currents or the projection of instrument collected data onto *vertical sections* by supersampling data points. For details on the complete activity set and usage, please see the standard oceanographic library included with Microsoft's Trident Workflow system [1], which was created as a direct result of this collaboration.

Based on these activities, we created a set of 12 visualization-based workflows that provide a representative cross-section of visualization workflows we observed or collected. These workflows each consisted of from 8 to 20 activities and comprised the test cases for our visualization architecture described in the next section. Each workflow loads the necessary inputs from a data store, transforms the input datasets into a new dataset, and then outputs the data to the COVE visualization engine to create a time series visualization of the data. These set of workflows are listed in Table 2 along with the primary scenarios they apply to and a broad measure of how data-intensive, computation-intensive, and visualization-intensive they were relative to the other workflows in the sample.

Table 2: Representative workflows tested based on ocean science scenarios

Workflow	Scenarios	Data	Computation	Visualization
Advect Particles	1,2,3,6,7	Medium	Medium	High
Combine Data	1,4,5,9	Low	Low	Low
Combine Models	1,2,3,6	High	Low	Medium
Compare Models	1,2,6	High	High	High
Compare Data to Model	1,2,6,7	Medium	Medium	Low
Filter Model	1,2,8	Medium	Low	Medium
PCA Projection	2,4	Medium	High	Medium
Regrid Model	1,2,7,8	Medium	Medium	Medium
Subsample Terrain	3,9	Medium	Low	High
Supersample Data	1,3,5	Medium	Low	Medium
Verify Model	2,6	High	Low	Medium
Vertical Section	1,5,6	Low	Low	High

The overall result of this effort suggests that there are no obvious or simple patterns in the workload of oceanographic analytics, and therefore no obvious or simple system that can be built to satisfy the requirements. Informed by this effort, we have designed a general platform for visual data analytics that spans these requirements, incorporating workflow, visualization, and cloud-based data access.

3.3 Cost Model

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. The *Data Store* component may be a remote query service: a database, an OPeNDAP server [22], or an Open Geospatial Consortium web service [23]. These services are typically outside the scope of a workflow system, though calls may be issued from the context of a workflow. We model the *Visualization* component as distinct from the *Workflow* component for two reasons: First, there are a variety of stand-alone visualization systems found in practice [6-8]. Second, the visualization step occurs last and benefits from access to a GPU, and is therefore often evaluated independently from the rest of the workflow. The *Client* component is responsible for processing user interaction and issuing calls to the upstream pipeline. This model allows us to seamlessly move the Visualization component from platform to platform based on current graphic processing needs; it can be tightly bound with the client for interactivity, tightly bound with the workflow system to minimize data transfer, or run independently to leverage external graphics resources.

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} that respects data flow order. For example, Fig. 1 illustrates a configuration that maps the Data Store to the Cloud, and the Trident Workflow Service, Visualization Engine, and COVE Client to the local computer.

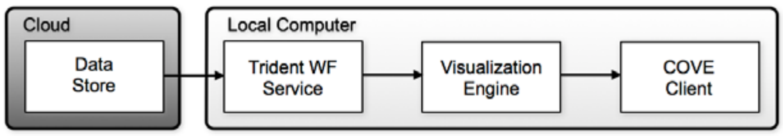

Fig. 1. An example of an architectural configuration mapping software components to resources. In this case the data is provisioned in the cloud and all other tasks are local.

We express the cost of each scenario as the sum of the workflow execution time, the visualization execution time, and the total data transfer cost between each pair of adjacent steps. That is:

COST = RAW_TX + WF_COMP + WF_TX + VIS_COMP + VIS_TX **(1)**

where

RAW_TX = RAW_SIZE / BANDWIDTH_DATA_WF
WF_COMP = WF_WORK (RAWSIZE) / PROCESSOR_WF
WF_TX = WF_SIZE (RAWSIZE) / BANDWIDTH_WF_VIZ
VIS_COMP = VIZ_WORK (WF_SIZE) / PROCESSOR_VIZ
VIS_TX = VIZ_SIZE (WF_SIZE) / BANDWIDTH_VIZ_CLIENT

RAW_SIZE is the size in bytes of the input dataset. BANDWIDTH_DATA_WF, BANDWIDTH_WF_VIZ, and BANDWIDTH_VIZ_CLIENT are the bandwidth between the data source/workflow system, the workflow system/visualization system, and the visualization system/client respectively. WF_SIZE and VIZ_SIZE are functions of the final output size based on the input data size for the pipeline step. PROCESSOR_WF and PROCESSOR_VIZ are the processor speeds for the respective machines, accounting for the potentially significant difference between server 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 [24]. These functions are typically polynomial in the size of the input data, but we find that even rough linear estimates of the workflows often provide a reasonable estimate.

Although this model captures the cost of the pipeline, it is not directly useful for prediction or optimization because the parameters are too difficult to estimate *a priori*. Therefore, we retain this model as a reasoning tool in Section 5, but experiment with a simpler proxy model based *only* on data transfer overhead. This proxy model, although simple, frequently captures the relative cost between different architecture configurations, as we will see. In this case, the model is

$$\text{COST} = \text{RAW\_TX} + \text{WF\_TX} + \text{VIS\_TX} \qquad \textbf{(2)}$$

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 over our workflow set, we leverage three existing systems: the COVE visualization system, the Microsoft Trident workflow system, and Microsoft Windows Azure cloud computing service. Communication between the components is provided through system I/O services if the components are co-located or by RESTful HTTP interfaces when distributed. Each component is 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 use, 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. COVE provides all the essential features of these commercial geo-browser systems, as well as enhancements designed in cooperation with ocean scientists.

In particular, COVE incorporates better support for the time and depth dimensions of ocean data sets. Visualizations can be animated and synchronized in time. COVE also provides extensive terrain visualization support, as each scientist may require a different set of terrain information. To provide more visual cues for the underwater terrain there are depth-based color gradients and contour lines as well as user adjustable shading and terrain detail to enhance visualization of seafloor features.

To enable experiment planning, asset deployment and tracking, and observatory design, enhanced interactive layout facilities are provided. To support vessel track routing 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.

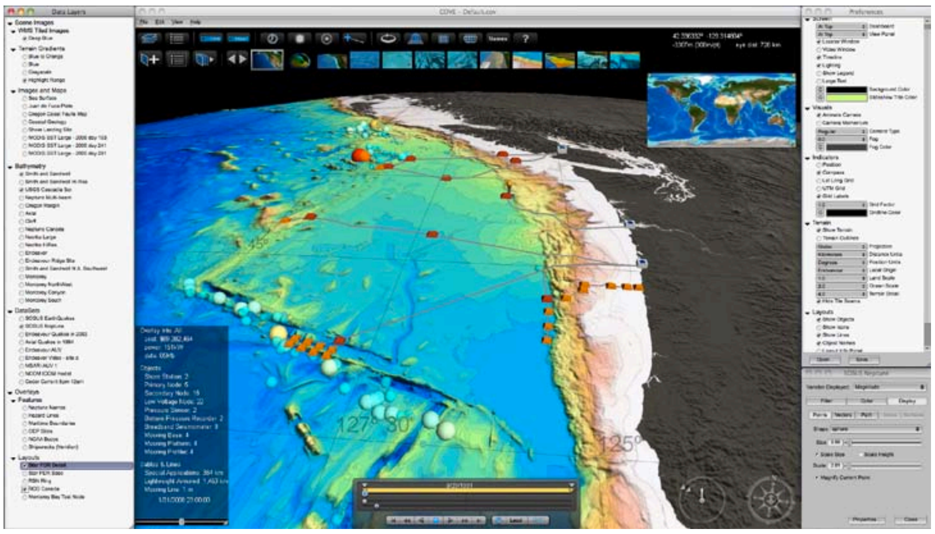

Figure 2: COVE displays a geo-positioned scientific data, seafloor terrain, images, and instrument layout with selectable layers on the left and rich visualization controls on the right.

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.

COVE has been successfully deployed for a variety of tasks. It was a key tool in the design of a planned deep-water ocean observatory 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 deployments, limitations became apparent. The geo-browser interface was empowering to novices, but expert users required extensibility for data manipulation. Also, as datasets grew 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

The Trident Workflow system, developed at Microsoft Research, is a domain-independent workbench for scientific workflow based on Microsoft's Windows Workflow Foundation. The system supports a high level component based view of scientific tasks that offers a number of advantages over traditional script-based approaches including visual programming, improved reusability, provenance, and execution in heterogeneous environments. In addition to these 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, task monitoring, and support for fault-tolerance and failure recovery.

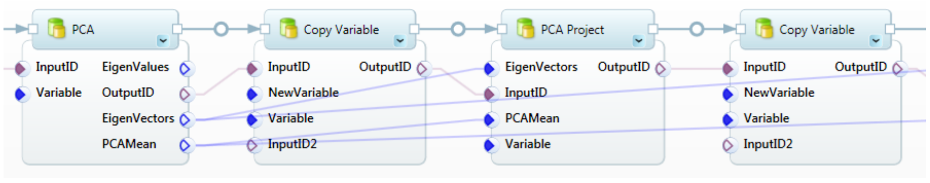

Fig. 3. This image displays an example of the interactive workflow editing interface of Trident.

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 (Fig. 3). Trident provides cross-platform support using Silverlight, a downloadable cross-browser, cross-platform, and cross-device plug-in for delivering .NET-based applications over the Web.

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 HTTP GET and POST calls. This is the communication interface used by COVE to provide cross-platform 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 for web applications 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 entirely responsible for outfitting the virtual machine with the needed software.

Windows Azure has three components: 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 the .NET API or the Win32 API. A Web Role package responds to user requests and may include an ASP.NET web application. A Worker Role, often 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 Worker Roles.

For persistent storage, Windows Azure provides three storage options: *Tables*, *Blobs*, and *Queues*, all accessed via a RESTful HTTP interface. A table is 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 offering standard relational storage based on SQL Server. In our system, we model data sources as Blobs and simply retrieve them for processing by our workflow engine.

4.4 Architecture Configurations

With these systems there are a variety of configurations that can be created to run the visualization workflows enumerated in Table 2. Fig. 4 illustrates the six architectures we evaluate.

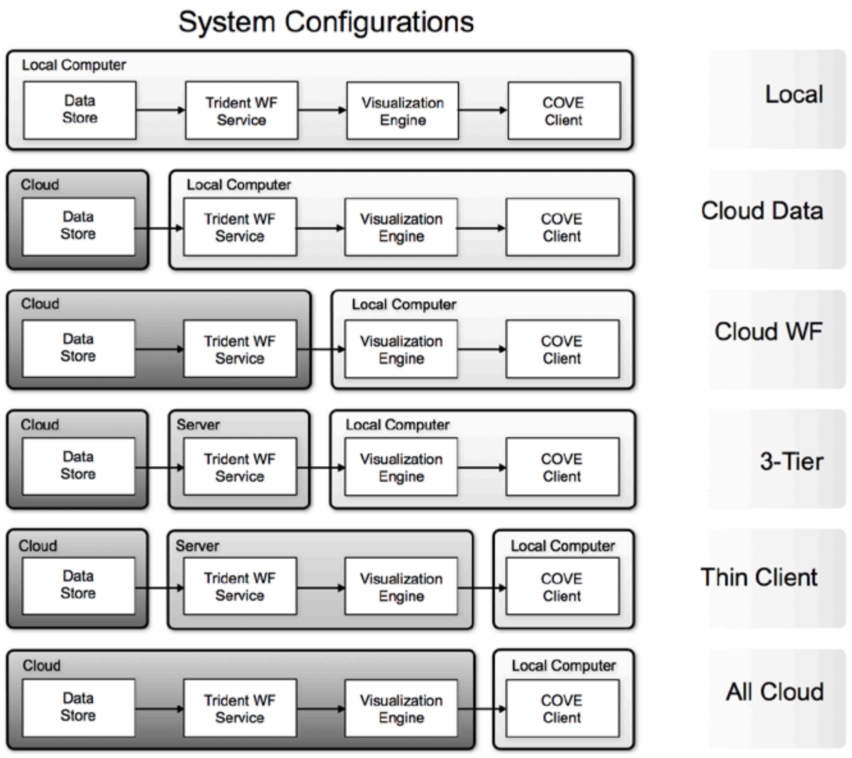

Fig. 4. The 6 evaluated configurations of the COVE + Trident + Azure system.

In the *Local* configuration, all the data and visualization is 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.

In *Cloud Data,* the data has been moved to the cloud. This configuration allows larger data sizes and also allows sharing of the data with other researchers, but with the overhead of downloading all necessary data to the local system.

In *Cloud Workflow,* the computation has been moved to the 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. However, this configuration still incurs the cost of downloading the filtered data.

In *3-Tier,* the computation has been moved remotely to a server and the data stored on in cloud storage. 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 to a remote location.

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.

Finally, the *All Cloud* configuration allows all the data to be handled in the cloud, with a minimum of network overhead since only the visual product is transferred over the network. The drawback is that the environment is usually unspecialized. In particular, the cloud typically does not provide graphics support for fast visualization.

4.5 Data Model

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

Files are downloaded by the workflow system from the data store and cached on the workflow system. The workflow system then accesses them from the local cache. This transfer mechanism could be optimized to reduce the overhead of local disk IO, but the local storage also allows for re-use of cached files in future workflows. Further, we observe in Section 5 that the local 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 through HTTP using a RESTful API. Although Trident provides access to SQL server for data storage, we found the current implementation for serializing and de-serializing 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. This means large files can exceed physical memory and lead to thrashing. For our Trident workflow activities, we instead use a lazy loading strategy. We load only a descriptive header when opening a file, and read in sections of the file on demand. This technique reduces the memory footprint and prevents thrashing.

The data files we access are taken directly from our collaborators. Each dataset is represented as a NetCDF [25] 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 NetCDF CF-Metadata naming conventions to standardize identification of position and time variables.

4.6 Programming Model

The Trident activities are written in C and deployed to a dynamic library for increased performance. The object interface for the library is then wrapped by a set of .NET managed C# Trident activities. Each activity typically accesses a single method in the library. This design also made it easy to expose the same functionality available to the workflow system to other systems. For example, since Trident was not yet available natively on the Windows Azure service, we created a substitute workflow shell on Azure that executed our workflow activities.

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. While the activities may execute either serially or in parallel according to the instructions in the workflow specification, all our workflows operate serially to simplify performance monitoring.

5 Experimental Analysis

We tested the 12 benchmark workflows in Table 2 on each of our 6 architectural configurations in Fig. 4 to record the 5 cost components of Eq. 1. Since the final dataset is too large to visualize effectively in this paper, we show a summary of the overall performance results in Fig. 7. This figure displays the average time of each of the cost components across the entire workflow set for each configuration.

Setup. We instrumented COVE and all of the Trident activities to record wall clock time for each of the components of the cost model: network transmission (RAW_TX, WF_TX, VIZ_TX), workflow computation (WF_COMP), and visualization creation (VIZ_COMP). We used the three systems described in Table 3 as the Local Machine, Web Server, and Azure Web Role in our architecture configurations.

Table 3: Physical Specification of Experimental Systems

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 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. 5, averaging around 150MB per task. Typical datasets include time steps of an ocean simulation, a set of "glider tracks" from an Autonomous Underwater Vehicle (AUV), or a terrain model for a region. All datasets pertain to the Pacific Northwest or Monterey Bay region and are "live" in the sense that they are actively used by scientists.

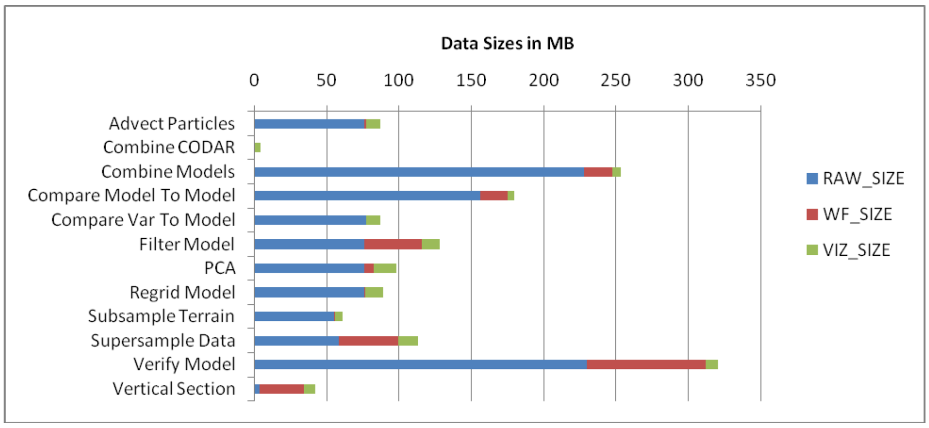

Fig. 5. Data sizes used in the experiments. Each bar is broken into three sections: the Raw data size (RAW_SIZE), the filtered data size generated by the workflow (WF_SIZE), and the size of the final result generated by the visualization (VIZ_SIZE).

Summary of Findings. 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 capture performance?

Our results show that: (1) There is no "one size fits all" architecture — the appropriate configuration depends on workflow characteristics (Fig. 7); (2) client-side processing is a crucial resource for performance, assuming data can be pre-staged locally to minimize transfer costs (Fig. 8); (3) access to a GPU strongly affects the performance of visual data analytics workflows, meaning that generic, virtualized cloud-based resources are not ideal (Fig. 9); (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 illustrates 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 of 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 ratio of data output to data input in the workflow or WF_RATIO = WF_SIZE / RAW_SIZE. In Fig. 6, the time profile for two workflows is 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 *Cloud WF*, where the data is processed on the same machine where it resides, however, when WF_RATIO > 1, the preferred configuration is *Cloud Data*, where data is sent to the local computer 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.

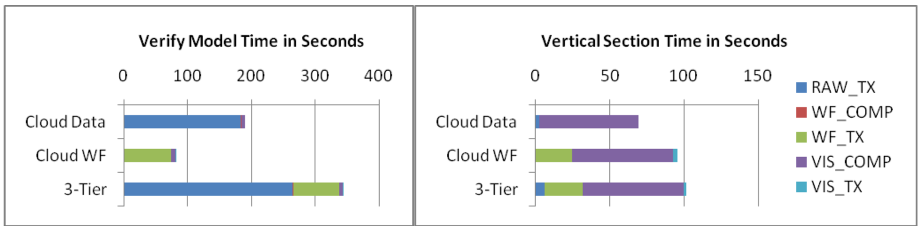

Fig. 6. Time profile comparison of a workflow with WF_RATIO < 1 on the left and WF_RATIO > 1 on the right. When WF_RATIO < 1, the preferred (non-local) strategy is to push the computation to the data using the *Cloud WF* configuration. When WF_RATIO > 1, the better strategy is to bring the data to the computation using the *Cloud Data* configuration.

5.2 Client-side Processing Improves Efficiency

At these modest data sizes, the local data configuration performed well in all cases. Fig. 7 shows the average performance across all benchmark workflows. The performance benefits are perhaps not surprising — desktop computers are increasingly powerful in terms of CPU speed and memory size, and are typically equipped with GPU to accelerate visualization. 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.

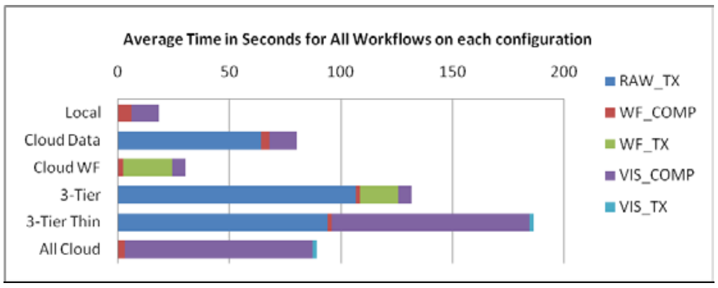

Fig. 7. The average runtime of all 12 workflows for each of the 6 architecture configurations is dominated by data transfer overhead. The *Local* configuration, although not necessarily feasible in practice, eliminates this overhead and therefore offers the best performance. This result suggests that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

5.3 Visual Analytics Benefit from GPU-Based Processing

Tasks involving significant visualization processing benefit from having access to GPUs. Although GPUs are becoming popular for general computation due to their vector processing capabilities, they can of course be used for visualization tasks with no change to the application. 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, in the *Thin Client* and *All Cloud* configurations, the visualization engine ran in the cloud and performed rendering in software using the Mesa 3D library, a state-of-the-art OpenGL software renderer. On average, the workflow set ran 5x faster overall with access to a GPU (Fig. 7). 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.

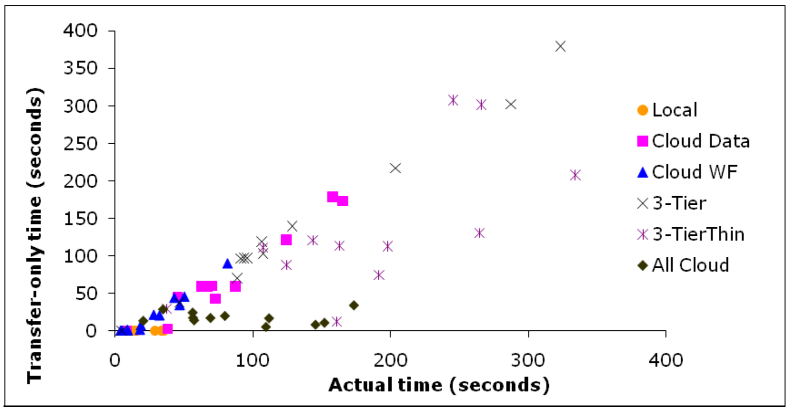

Fig. 8. Scatter plot of estimated results from Eq. 2 compared to actual results.

5.4 A Simple Cost Model Informs Architecture Decisions

The proxy cost model presented in Section 3 is very simple, capturing only the data transfer costs between each step. 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 several of the computations, suggesting that a very simple “architecture optimizer” could be based on it.

In Fig. 8, we plot the estimated running time against the actual measured times using a model that ignores everything except for transfer times. A linear relationship is clear, though of course it underestimates CPU and Visualization-heavy workflows, as well as fully local configurations that do not require any data transfer. For these cases, we are exploring an incrementally more sophisticated model based on parameters that can be estimated by the scientists ad hoc.

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 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 (Cluster Exploratory Award # 0844572 and MREFC Award # 1005697, through a subcontract from the Ocean Observatories Initiative through Woods Hole Oceanographic Initiative [26]).

References

[1] Barga, R. S., Jackson, J., Araujo, N., Guo, D., Gautam, N., Grochow, K., Lazowska, E.: Trident: Scientific Workflow Workbench for Oceanography. In: IEEE Congress on Services, pp. 465-466 (2008)
[2] Bavoil, L., Callahan, S. P., Scheidegger, C. E., Vo, H. T., Crossno, P. J., Silva, C. T., Freire, J.: VisTrails: Enabling Interactive Multiple-View Visualizations. In: IEEE Symposium on Visualization, p. 18 (2005)
[3] Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C., Katz, D. S.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming. 13, pp. 219-237 (2005)
[4] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience. 18, pp. 1039-1065 (2006)
[5] The Triana Project, http://www.trianacode.org
[6] MayaVi, http://mayavi.sourceforge.net/
[7] ParaView, http://www.paraview.org/
[8] The Visualization Toolkit, http://www.vtk.org

[9] Grochow, K., Stormer, M., Kelley, D., Delaney, J., Lazowska, E.: COVE: A Visual Environment for ocean observatory design. Physics Conference Series. 125, pp. 012092-012098 (2008)
[10] Windows Azure Platform, http://www.microsoft.com/windowsazure/
[11] McCormick, B. H., DeFanti, T. A., Brown, M. D.: Visualization in Scientific Computing. Computer Graphics. 21, (1987)
[12] Abram, G., Treinish, L.: An Extended Data-Flow Architecture for Data Analysis and Visualization. In: IEEE Symposium on Visualization, p. 263 (1995)
[13] Bethel, E. W., Campbell, S., Dart, E., Shalf, J., Stockinger, K., Wu, K.: High Performance Visualization Using Query-Driven Visualization and Analytics. Lawrence Berkeley National Laboratory. (2009)
[14] Ma, K.-L., Wang, C., Yu, H., Moreland, K., Huang, J., Ross, R.: Next-Generation Visualization Technologies: Enabling Discoveries at Extreme Scale. SciDAC Review. pp. 12-21 (2009)
[15] Brodlie, K., Duce, D., Gallop, J., Sagar, M., Walton, J., Wood, J., "Visualization in Grid Computing Environments," presented at the Proceedings of the conference on Visualization '04, 2004.
[16] Wu, Q., Gao, J., Zhu, M., Rao, N. S. V., Huang, J., Iyengar, S.: Self-Adaptive Configuration of Visualization Pipeline Over Wide-Area Networks. IEEE Trans. Comput. 57, pp. 55-68 (2008)
[17] Bright, L., Maier, D.: Efficient scheduling and execution of scientific workflow tasks. In: Scientific and Statistical Database Management, pp. 65-74 (2005)
[18] Langguth, C., Ranaldi, P., Schuldt, H.: Towards Quality of Service in Scientific Workflows by Using Advance Resource Reservations. IEEE Congress on Services. 0, pp. 251-258 (2009)
[19] Wu, Q., Gu, Y., Bao, L., Jia, W., Dai, H., Chen, P.: Optimizing Distributed Execution of WS-BPEL Processes in Heterogeneous Computing Environments. Quality of Service in Heterogeneous Networks. pp. 770-784 (2009)
[20] Monterey Bay Aquarium Research Institute, http://www.mbari.org
[21] College of Ocean and Fishery Sciences, University of Washington, http://www.cofs.washington.edu/
[22] OPeNDAP: Open-source Project for a Network Data Access Protocol, http://opendap.org/
[23] OGC Web Map Service, http://portal.opengeospatial.org/standards/wms
[24] Boulos, J., Ono, K.: Cost estimation of user-defined methods in object-relational database systems. ACM SIGMOD Record. 28, pp. 22-28 (1999)
[25] Jenter, H., Signell, R.: NetCDF: A public-domain software solution to data-access problems for numerical modelers. Preprints of the American Society of Civil Engineers Conference on Estuarine and Coastal Modeling. p. 72 (1992)
[26] Woods Hole Oceanographic Institution Submersibles, http://www.whoi.edu/