# Versioning for Workflow Evolution

Eran Chinthaka Withana, Beth Plale
School of Informatics and Computing
Indiana University
Bloomington, Indiana

{echintha,plale}@cs.indiana.edu

Roger Barga, Nelson Araujo
Microsoft Research,
Microsoft Corporation
Redmond, Washington

{barga,nelsona}@microsoft.com

## ABSTRACT

Scientists working in eScience environments often use workflows to carry out their computations. Since the workflows evolve as the research itself evolves, these workflows can be a tool for tracking the evolution of the research. Scientists can trace their research and associated results through time or even go back in time to a previous stage and fork to a new branch of research. In this paper we introduce the workflow evolution framework (EVF), which is demonstrated through implementation in the Trident workflow workbench. The primary contribution of the EVF is efficient management of knowledge associated with workflow evolution. Since we believe evolution can be used for workflow attribution, our framework will motivate researchers to share their workflows and get the credit for their contributions.

## Categories and Subject Descriptors

D.2.6 [**Programming Environments**]: Programmer Workbench; D.2.6 [**Distribution, Maintenance, and Enhancement**]: Version Control; E.2 [**Data Storage Representations**]: Object Representation;

## General Terms

Algorithms, Management

## Keywords

Workflows, Evolution, Versioning

## 1. INTRODUCTION

Computational science experiments often involve a sequence of activities to be carried out, with a set of configurable parameters and input data, producing outputs to be analyzed and evaluated further. Depending on these outputs, scientists will tweak input parameters, input data, and activities of the experiments and its flow, to improve experiment results. If experiment activities or parts of the experiment can be automated, researchers utilize workflows to automate repeatable task steps in an efficient manner. In a workflow setting, rather than doing everything manually, a scientist will encode their algorithms and experimental procedures as workflows and use the flexibility, tools and features of workflow engines. When a workflow framework is used continuously over an extended period of time, the research will likely evolve along different dimensions which will affect and evolve the associated workflow(s) as well. After a period of time a researcher may need to review what they have done for a variety of reasons, possibly visiting results from weeks or months ago. Even in operational settings, where workflows are used to produce daily results such as data cleaning and loading, operational workflows will periodically change. We have identified through discussion with users several reasons why a researcher may be interested in the evolution of his or her experimental history. For one, they may want to visualize the evolution of the research to see the path to the current state and what the previous attempts were. Also a scientist may backtrack to a previous stage and take a new direction. There may be errors in an algorithm or the experiment and want to trace back to the origin of that error. Or they might want to get a list of the data products and results affected by this error. Scientists might want to visualize the data products their experiments produced over the time and use them for various evaluation purposes. Finally, in addition to tracing the workflows over time, scientists may also be interested in re-producing workflows.

Workflows encapsulate a vast amount of knowledge associated with scientific experiments. We believe tracking the evolution of workflows will help to aggregate this knowledge for later analysis. The benefits of such a system include the following:

1. **Tracking effects over Time.** When scientists associate their research with workflows, tracking the evolution of these workflows becomes an approximation to the initial problem of tracking the evolution of their research. Along the evolution of a workflow, all the components within it will also evolve. Scientists should be able to look at the result of a workflow execution and reason about how the research came to current level to produce that particular output. Another important aspect of tracking the evolution is to track the lineage and the roots of errors in the experiment. For example, if an error is found in an algorithm or an input to the workflow, the evolution information should be helpful for the scientists to track back in time to find the root of the error or the affected experiments due to these errors. This information can be useful to revert back to the



This work was performed while first author was an intern at Microsoft Research and is funded in part by NSF Grant CNS-0420580

last known good configuration and then start research from that point onwards.

2. **Comparing results.** A given research line might evolve in more than one direction. It is really important to understand the changes on these directions by comparing the difference between the outputs of two or more versions of the same research. For example, given two outputs of two different versions of a workflow, one should be able to deduce the reason for the difference between the two results by looking at the lineage information.
3. **Attribution.** When a workflow is executed, attribution information such as who performed the experiment, who owns or created the workflow, who owns the data products, etc., can be gathered. This attribution information will later be useful to track down the issues or to give proper credit to the original owners. Also, while carrying out experiments it is becoming more common to reuse subset graphs within a workflow Scientists can utilize not only the algorithms and implementations developed by others, but also the data products generated including optimally derived model parameter configurations. For example, in natural language processing, researchers will use bilingual corpora published by standard bodies to test their algorithms. This reuse of a public corpus will reduce effort but will increase its acceptance. In research, it is not only the technical aspects that will matter; sharing and attribution of research can and should be an integral part of research. We can access and download subset graphs from sources like myexperiment.org [17] to reduce development costs. Tracking this kind of contribution within our evolution framework will not only provide a way to track contributions, but also to track attribution for proper accreditation to the contributors. This last point we believe is an important aspects of scientific research from social point of view. At the same time, scientists in some domains are already motivated to publish their work and like to see their work being appreciated and attributed properly. We believe a workflow evolution framework should also support work attribution. If a workflow uses work from other research, current workflow should have a way of attributing to previous work.

In this paper we introduce the Workflow Evolution Framework (EVF) and versioning model to help scientists manage knowledge encoded in their workflow executions. We show that versioning can be done efficiently. We discuss its implementation and use for tracking changes to the images in a Microsoft Word document.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 introduces a couple use cases that have motivated our research and Section 4 explains the conceptual model which enables knowledge management in workflow evolution through versioning. Section 5 discusses the architecture of our system and Section 6 evaluates several versioning options. Section 7 concludes with a summary and discussion of future work.

## 2. RELATED WORK

Numerous workflow orchestration and composition tools exist [18][4][11][12][7]. The tools provide different feature sets and selection of a given tool often depends on the usability of a tool in a user's particular domain. Workflows have become so useful that they have become part of almost every major e-Science platform [13][14]. Workflows define the machinery for coordinating execution of scientific services and linking together the resources involved in an experiment. Workflows also help the scientists encode repetitive tasks enabling them to focus on the science. Once created, most workflows can be shared with others, which helps establish best practices, but also improves the productivity of the entire research community.

The information model of workflow evolution shares a lot in common with provenance collection frameworks [29][14]. Both capture a task graph and can contain information about the environment in which the task graph will or has executed. A significant difference between the two is that workflows are a plan where provenance is a record of execution [28]. Workflow information is gathered in advance of a run; provenance information is collected on the fly sometimes with and sometimes without the benefit of workflow information. Information collected using the proposed workflow evolution framework can complement provenance collection. The workflow evolution framework can be a value to a provenance system as it uses a single form to represent the workflow so can be mined more easily than can workflow scripts which are babel of formats and languages. Since most workflow frameworks require explicit invocation of provenance handlers to capture provenance, researchers argue for the automatic generation of provenance data by workflow enactment engines that can be managed through underlying storage services [6].

Researchers are interested in lineage information because this information is important to properly document the scientific experiments [8]. The Earth System Science Workbench [16] uses lineage information to detect errors and determine the quality of the data sets. The CMCS [24] system uses lineage information to establish the pedigree of the datasets they were using. Workflow evolution itself has been studied. VisTrails [15] for example provides functionality to capture and track workflow evolution. The tool also provides a workflow orchestration environment for visualization experts to compose workflows. The data model[30] of Vistrails captures steps in the creation and exploration of visualizations, whereas our model takes a data centric approach, capturing the artifacts in an experiment and the relationships between them. Also to our knowledge VisTrails does not support attribution such as we do through tracking the contributions to workflows. Casati [9] introduces the dynamic aspects of workflow evolution within a workflow engine with emphasis on the complexities of evolving workflows when under the condition of running instances. We examine the problem in a slightly more abstract sense and limit our examination to static workflows. We define workflow evolution over an extended time duration and are not limited to the runtime of an average workflow.

Versioning has been applied at the application level[26], the file system/database level[10] and at the disk storage level[14]. For efficient implementation of versioning, a system must identify the objects to be stored, and consider the methods to store, represent and retrieve versioned objects. Systems have saved an object (or a copy of the object) as a version[20][21][26][28],

using delta computation[3][19][22] for versioning objects within the systems. The Elephant file system[26], has a novel versioning strategy positing that different types of files need different versioning strategies. In our system, as we demonstrate in the performance evaluation, there are advantages to supporting multiple versioning techniques. Different models of versioning systems have been proposed, depending particularly on the requirements of the system that they will be used. For example, S4[19] focuses on securing the versioned objects and Sprite LFS[25]focuses on lowering the disk access overhead for small writes. We focus on optimal versioning within an eScience, workflow driven setting.

# 3. USE CASES

There are two use cases that have been particularly useful in motivating our research. We discuss them here.

## 3.1 Supporting Research Reproduction

Research papers can contain numerous graphs, charts, and images, but it can be difficult to determine the lineage of a figure, particularly once time has passed and it is a colleague who is interested. The reasons for this are several, 1.) The artifacts associated with the figure including metadata about the versions and locations of inputs, parameters, workflows, etc. are not recorded or tracked, 2.) If the information is available, collecting the data and getting it into an executable state is difficult for a third party who wants to reproduce results, and finally 3.) If there is a change to parameters or any other artifact, the interested party will have to manually run the experiments, copy the results and re-insert into the paper.

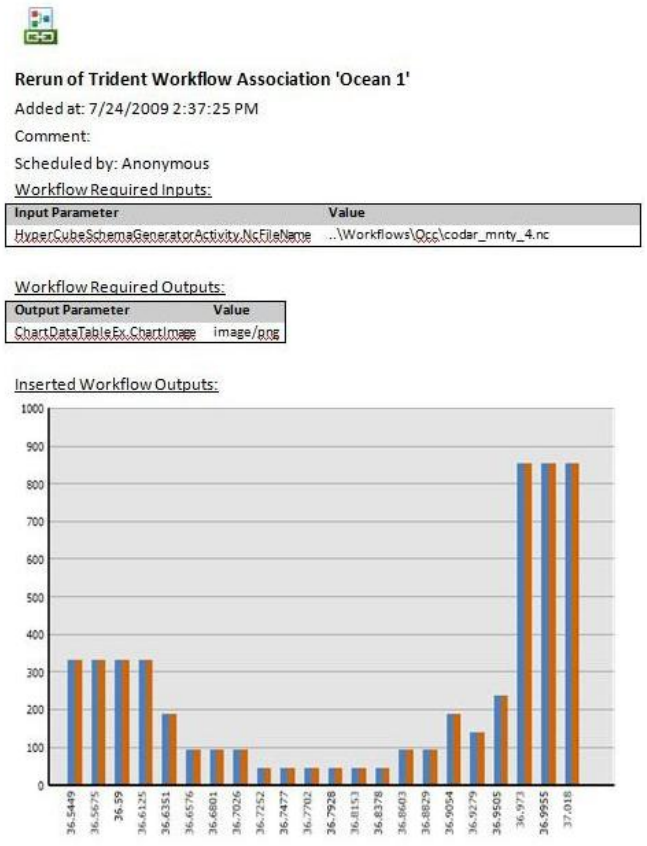


**Figure 1: Embedded Reproducible Workflow with Output**

If instead the workflow and metadata of a final graph or image are embedded into the research paper itself, and if a framework exists to regenerate the graph, it introduces functionality not previously available will be an important value addition to both the readers and the authors of the paper. To enable reproducing of research, we implemented a Microsoft Word® plug-in, shown in Figure 1 as an aid to users to embed regeneration workflows into their research manuscripts. Once included, user can insert the outputs of these workflows, like visualizations of the results, into the Word document. Figure 1 shows an embedded workflow inside a word document, together with the output visualization of the workflow and meta-data to re-generate the workflow. At a later time, the reader or author himself or herself can re-execute the embedded workflow to re-produce the results. The plug-in enables a user to re-run the embedded workflow on a Trident server. The add-in can be used to organize jobs, results, and workflow from various Trident servers, helping to more easily manage research.

## 3.2 Scientific Workflows

Linked Environments for Atmospheric Discovery (LEAD) [13] pioneered cyber-infrastructure for adaptivity in response to the immediate needs of understanding emerging severe storm patterns. It developed a workflow system, later based on Apache ODE, and a data subsystem that enabled workflows to bind to the latest atmospheric observational data. Many of the workflows preferred by users included the Weather Research Forecast model (WRF) [23], which is configured to generate a forecast that is valid anywhere from 3 hours to 4 days. The models require a complex set of inter-connected parameters (stored as Fortran "namelist" files) as input to properly initialize the model. Once a model is run, a researcher may revisit the namelist file to achieve more optimal performance for instance. Our proposed framework will help the researcher understand the lineage of the workflow, the data and parameters related to a given output result, and the sources of data. If the researcher needs to change a set of parameters, the framework should enable him to go back in time, pick an old version, and create a new branch to carry experimentation in a new direction. These use cases motivate the overall EVF framework. In the next section we introduce the versioning model underlying EVF.

# 4. VERSIONING MODEL

## 4.1 Model

For workflow-based research to be reproduceable, a versioning strategy needs to consider the workflow and the associated data products, parameters, configurations and executable should exist, and be bound together. The Trident Workflow Evolution framework, where we implemented EVF, supports reproducibility by persisting all information about previously executed experiments. If the underlying data management services enables accessing of versioned data products, then using EVF a scientist can re-run previously executed experiments.

This versioning model (Figure 2) is built on two orthogonal dimensions of workflow evolution, namely direct *evolution* and *contributions*. *Direct evolution* occurs when a user of the workflow performs one of the following actions:

1. Changes the flow and arrangements of the components within the system
2. Changes the components within the workflow

3. Changes inputs and/or output parameters or configuration parameters to different components within the workflow

For example, a scientist might change the implementation detail of an algorithm used within the system, and add that component to the workflow, removing the previous one. *Direct evolution* will primarily come from a researcher's direct involvement in the research that is being tracked. On the other hand *Contributions* will track components that are reused from previous system. For example, a scientist may extract a BLAST processing module from an existing workflow available on the Web and use it within his workflow. Or he may create a new branch from the current research and take the research to a new direction. In either case, the research he was doing to that point will be a contribution to the new branch created.

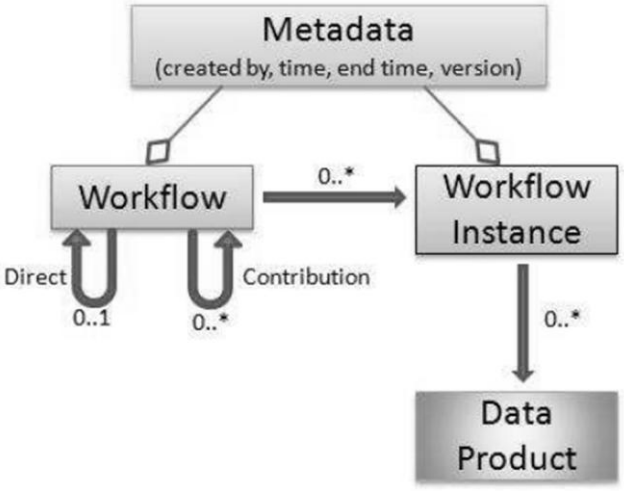


**Figure 2: Versioning Data Model within EVF**

One of the unique features in EVF is that it tracks both "direct evolution" and "contributions" to research. Together this contributes towards the existing eco-systems to acknowledge each other's contributions to the existing research and also encourages scientists to share and use existing work. Versioning of workflows and related artifacts is done at three separate stages of execution.

1. User explicitly saves the workflow
2. User closes the workflow editor
3. Executing a workflow in the editor: since workflow instances should always be associated with a workflow, EVF requires all the workflows to be saved and versioned before executing them.

This level of granularity will not capture all minor edits to a workflow, but in applying EVF in the use cases discussed in Section 3has established a level of sufficiency for this level of versioning for later retrieval and workflow evolution. Figure 2 gives the data model used for versioning of objects and the relationships between them. This model is designed such that all the artifacts related to an experiment can be captured and versioned using this model. Features of the model include the following:

- Each workflow a user creates and the execution of that workflow is recorded inside the system, together with the associated meta-data containing information like who own/ran the experiment, the time frames, validity period, etc.,
- Each workflow execution is associated with the workflow template used to run the experiment.
- All the data products used and generated in a workflow execution is associated with the corresponding workflow instance, enabling to track them back later.
- Each workflow has links to the direct evolution (unless it is the first workflow in the evolution), which will point to the next version of the workflow, if any, and to the contributions. These contributions track the previous work this workflow is using inside it attributing to the previous work.

All information is persisted in a registry, such as the Trident Registry. A new version of a workflow will be saved within the registry creating the next version when the user explicitly decides to save the workflow. But information about workflow instances and data products will be saved automatically.

The registry implementation must support the security and privacy of the information stored in it. In our implementation, the registry enforces a user role based authentication and authorization scheme to enforce the security and privacy information. But the user retains the ability to share his information among selected set of users or all the users.

# 5. ARCHITECTURE

Before discussing the details of implementation, we introduce the Trident research platform used to evaluate EVF.

## 5.1 Trident Research Platform

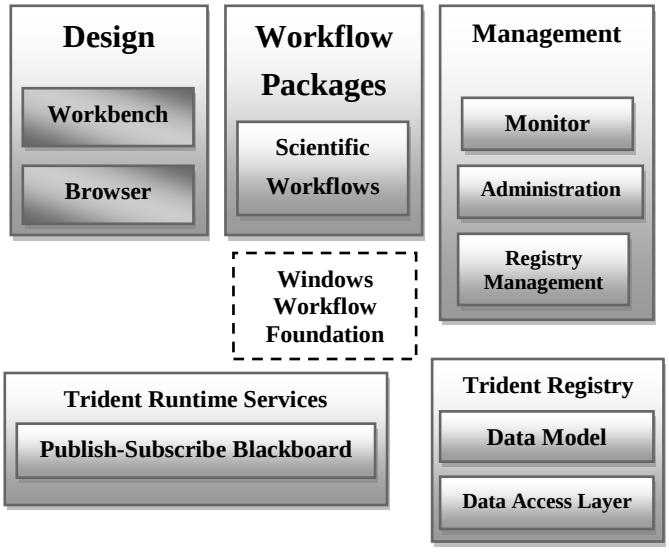


**Figure 3: Key Elements of Trident Architecture**

The aim in designing Trident has been to leverage existing functionality of a commercial workflow management system to the extent possible and focus the development efforts only on functionality required to support scientific workflows. The result is a smaller code base to maintain going forward, improving sustainability and manageability of the project, and an improved understanding of requirements unique to scientific

workflow. Trident is implemented on top of Windows Workflow (WF) [5], a workflow enactment engine included in the Windows operating system. The Windows WF extensible development model enables the creation of domain specific activities which can then be used to compose workflows that are useful and understandable by domain scientists. The key elements of the Trident architecture (shown in Figure **3)** include a visual composer and library that enable scientists to visually author a workflow using a catalog of existing activities and complete workflows.

The Trident registry serves as a catalog of known data sets, services, workflows and activities, and compute resources, as well as maintains state for all active workflows. It also enables searching for artifacts cataloged in it. With the use of data providers, it enables the integration of new data sources into Trident. These data providers can also be used to import or convert data from different formats to be used inside workflows.

An execution engine exists in Trident that supports launching workflows remotely and according to a schedule. Administration tools allow users to register and manage computational resources, publish workflows for external use, and track all workflows currently running or recently completed. Users can schedule and queue workflow execution based on time, resource availability, etc. A set of community tools includes a web service that enables users to launch workflows from any web browser and a repository that facilitates the publishing and sharing of workflows and workflow results with other scientists which integrates with myExperiment.org [17]. At the lowest level, Trident has a data access layer that abstracts the actual storage service that is in use from the running workflows. The data access layer is extensible and currently Trident supports a default XML store and SQL Server for local storage, and Amazon S3 [1] and SQL Server Data Services (SSDS) [2] for cloud storage.

Workflow Foundation provides several runtime services which can be used as required by attaching the service implementation to the workflow runtime. Two of the most useful for our implementation of Trident are a *tracking service* which enables event based tracking of a running workflow through the use of extensible tracking profiles, and a *persistence service* which allows the workflow executor to serialize and restore the entire working state of an in-progress workflow, allowing the executor to pause and resume workflows and archive intermediate state to any capable storage device.

## 5.2 Versioning Architecture

We implemented the Trident Workflow Evolution Framework (EVF) within the Trident Workflow Workbench [7] to demonstrate practicality of achieving the objectives of versioning efficiency. The versioning model of EVF is integrated into the Trident data model as shown in (Figure 4). This architecture can version files locally or can use the versioning capabilities of an external local or remote versioning system. After the integration, any object saved in to Trident registry is automatically versioned and can be retrieved later. Meta-data stored inside the framework enables the retrieval of any version of a given object. Figure 5 demonstrates the changed object view within Trident registry, enabling the access of versioned objects. In the example shown, in Figure 5, the ocean workflow has four different versions. Three versions refer to the versions in the "Direct Evolution" and Ocean Branch workflow refers to a branch created in the evolution of the workflow.

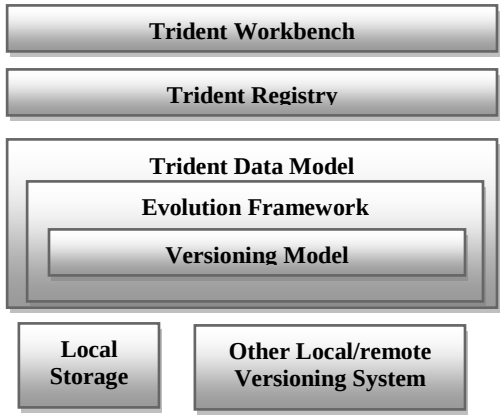


**Figure 4: Trident Evolution Framework Architecture**

### *5.2.1 Implementation*

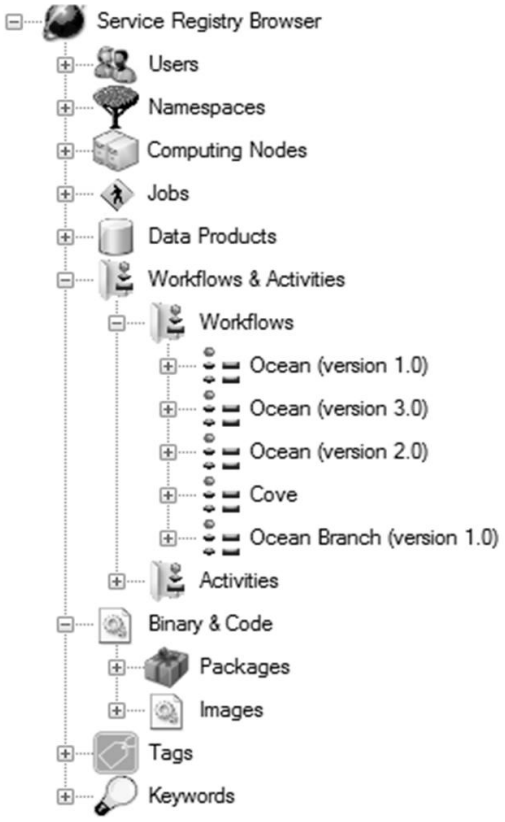


**Figure 5: Versioned Ocean View Workflows in Service Registry**

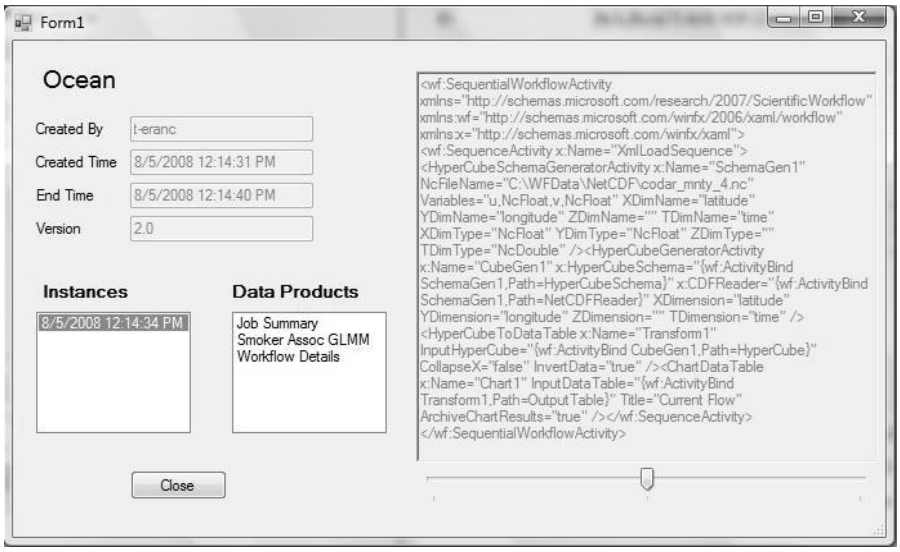


**Figure 6: The Workflow Evolution View**

In this section we will explain the implementation of the concepts of our EVF framework within the Trident Workflow Workbench [7].

Scientists use Trident Workflow Composition and Execution environments to create, edit and execute workflows. EVF is integrated into both Trident composition environments and to the service registry. Once the user is done creating a new workflow or modifying an existing workflow, he is required to save it to the service registry. A user can also retrieve an existing workflow from the registry. These retrieved workflows can be ones he created or can be download from Web resources like myexperiment.com. Once the scientist has the workflow, he will then execute it within Trident workflow execution environment. All the data products and execution variables will be tracked and automatically stored within service registry. Evolution of workflows will be recorded and can be viewed from the service registry. Once the user goes into the service registry, he will see all the workflows categorized by the name of the workflow.

Figure 5 shows the view of Trident Registry, showing the evolution of an Oceanography workflow. There are two versions created so far and also research is progressing along a different direction with "Ocean Branch" workflow. To check the workflow evolution, user will select the intended workflow and asks for workflow evolution. Figure 6 shows the timeline view implemented within EVF. Our framework will display,

1. A time line view of the workflow
2. Meta data for each and every version of the workflow, containing information on who created, the time duration it was active and the version
3. For each of the version, user will see the workflow instances created using it and the data products consumed and generated within them. User will have the options of visualizing the data products, stored as images, generated within them
4. Related contributions associated for each and every workflow and the direct evolution information

Time-line view (shown in 6) enables navigation, through time, by moving the time slider back and forward. This view also enables see all results that a particular workflow version created, along with the ability to select a result and track back to the workflow version that created it.

## 5.3 Architectural Features

The architectural features enabling the workflow evolution for the management of the knowledge associated with workflow executions and management are several, and are provided in more detail below.

**Unique Association of Research Artifacts to Workflows:** Once a workflow is executed, it is very important to associate the relevant data, parameters, configuration information and also the meta-data capturing information on who performed the experiment, when and where the output was saved, etc.,. In addition to these information related to the current instance of the workflow, we also need to keep track of the lineage of the workflow itself. These unique associations will not only help to manage the knowledge associated with the workflow, but also will help to re-produce the same research at a later time. Within the EVF framework, we enable these unique associations by recording this information inside our information model.

**Automatic Versioning:** EVF helps scientists to version workflows as and when they edit them. This enables a researcher to later retrieve a previous workflow for viewing or to create new branches from the previous workflows to take them in a new direction. Versioning of the workflow templates inside EVF is comparable to a typical version control system, but EVF also has the ability to work with other versioning systems to support different versions of the data products. EVF provides clearly defined extension points to add new versioning systems. Once a data provider, capable of versioning data products, is registered with the system, EVF will save enough information to retrieve a given version of a data product. When EVF is associating data products with workflow executions, it will also include this versioning information, so that the correct version of the data product can be retrieved later, in case the scientist is interested in reproducing the research. Also if an extension is

registered to handle the versioning system, EVF will use that extension to automatically retrieve the data and to execute the workflow within Trident. During workflow authoring process, scientist will keep on changing a workflow and might also save all the intermediate steps. But at the end he might only execute the last version of the workflow. Should the scientist opt to delete the previous versions, EVF gives the control to the user to select the versions to persist inside the registry or to remove from the system. This will not only reduce the clutter in the scientist's workspace, but also optimizes the workflow lineage information persistence.

**Validity of a Workflow: Versioning** of workflow brings up questions about the validity of a workflow. When a new version of the workflow is created, it is a questionable whether to leave the previous versions of the workflow to be still executable or not. Within EVF we leave this decision up to scientist and select one of the solutions. In our framework each workflow is assigned a unique ID and a version number VN and a time interval $[V^b, V^e)$ that represents the time interval during which a workflow was valid. A new version of workflow ID at time t is assigned a unique version number VN, and validity interval $[V^t, \infty]$. If only one version of a workflow is allowed to be active at any point in time, which is a configuration option, the previous version VN is assigned a validity interval of $[V^0, V^t)$. Validity of a workflow only restricts whether it can be re-executed or not. All the information of previous workflows can still be tracked irrespective of this option.

**Navigation through Time: Navigating** through time can give a researcher a unique view on the evolution of their research. A scientist might see change or improvement in the results of their experiments over time. They may witness the effects of the different data sets being used, or may use visual evolution to determine ownership of a piece of work or to see the contribution this particular piece of research has from the previous or related work. With the information model (Figure 2) we propose in our work, navigation through time becomes easier because it captures all the information needed for the scientists to visualize the evolution of their work. Since this information model associates the workflow instances of a given version of the workflow, scientists can also see the runtime information of each and every workflow execution. We believe that providing this information along a time-line will give users more insight into their research.

# 6. Performance Evaluation

The performance evaluation of the versioning model is focused on evaluating three strategies for versioning. These approaches have performance and usability tradeoffs that we attempt to capture. The three strategies to be used during versioning are.

1. No Delta, No Checkpointing: each version of the workflow is saved as it is to the file system and differentiated by its version number.
2. With Delta, No Checkpointing: the difference between the current version and the previous version (delta) of the workflow is saved in to the file system. If a version has to be recovered, all the deltas up to that version, must be applied to the first version of the workflow
3. With Delta, With Checkpointing: the difference between the current version and the previous version (delta) of the workflow is saved in to the file system. But the full workflow is saved after each fixed number of versions (checkpoints). To recover a given version, the closest Checkpointed workflow should be retrieved and the deltas after that point should be applied to that workflow.

We identify two workflows, arbitrarily called “O” and “M”, selected for their difference in size of the workflow and on the difference in bytes between two successive versions of the workflows; see Table 1. The O workflow use case is used to analyze the overheads of working objects, which are subjected to small changes, compared to the size of the object. The M workflow use case analyses the overheads of working with objects which will be subjected to larger changes. All the evaluations are run inside a 2.0GHz dual-core processor, 4GB memory and on Windows 7 Ultimate 64-bit operating system.

| Workflow | Size (Bytes) | Delta (Bytes) |
|---|---|---|
| O | 1032 | 210 |
| M | 4087 | 2564 |

**Table 1: Workflows Used for Evaluations**

## 6.1 File Write

The first experiment evaluates the write time for successive versions of a given file, for each of the three different versioning strategies. Figures 7 and 8 show the variation of the time spent for writing the file against the version number of the file, for O and M workflows respectively. For both the workflows, "No delta, No Checkpointing" option performs better, because it takes constant amount of time to write a file as it is. "With Delta, No Checkpointing" case is worst, because before writing the new version of a file, it needs to recover the previous version and get the difference with that version. For this, the system needs to recover the previous version by retrieving all the previous versions. As proven in the graph the time to write a given version is directly proportional to the version number of the file. "With Delta, With Checkpointing" case, performs better than "With Delta, No Checkpointing" but not better than “No delta, No Checkpointing". For the larger M workflow, with larger deltas, "No delta, No Checkpointing" option performs at least 20-30 times faster than the other two options.

## 6.2 Version Recovery Time

This experiment evaluates the time taken to retrieve a given version of a file, for each of the three different versioning strategies. Figure 9 and Figure shows the variation of the time spent for recovering a version against the version number of the file, for O and M workflows respectively. For both the workflows, "No delta, No Checkpointing" options performs better, because to recover a version, it only needs to retrieve the file from the file system. "With Delta, No Checkpointing" case is worst, because to recover a given version, it needs time to retrieve all the previous versions to re-construct the file. As proven in the graph the time to retrieve a given version is directly proportional to the version number of the file. The compromise case "With Delta, With Checkpointing", performs better than "With Delta, No Checkpointing" but not better than “No delta, No Checkpointing". For both the workflows, "No delta, No Checkpointing" option performs at most 10 times faster than the other two options.

## 6.3 Storage Requirements

This experiment evaluates the storage requirements for each of the three different versioning strategies. Figure 11 and Figure 12

shows the accumulated storage requirement to save all the versions up to a given version against the version number, for O and M workflows respectively. As can be seen in both the graphs, "No delta, No Checkpointing" option takes the most storage to save a version of a file, and increases linearly with the increase of versions. For O workflow, the other two options uses 4-5 times less storage (at most) than "No delta, No Checkpointing" option. But for the M workflow, the storage saving is 2 times at most.

## 6.4 Amount of Data Retrieved to Recover a Version

This experiment evaluates the amount of data to be retrieved to recover a file version (recovery overhead) for each of the three different versioning strategies. Figure 13 and 14 visualize the recovery overhead of all the versions up to a given version against the version number, for O and M workflows respectively.

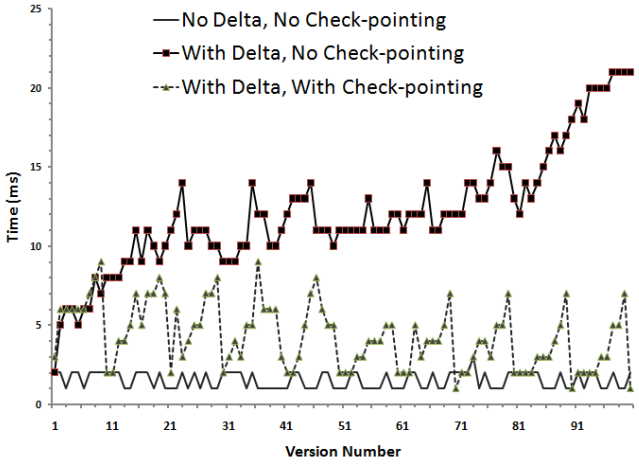


**Figure 7: File Write Time - O Workflow**

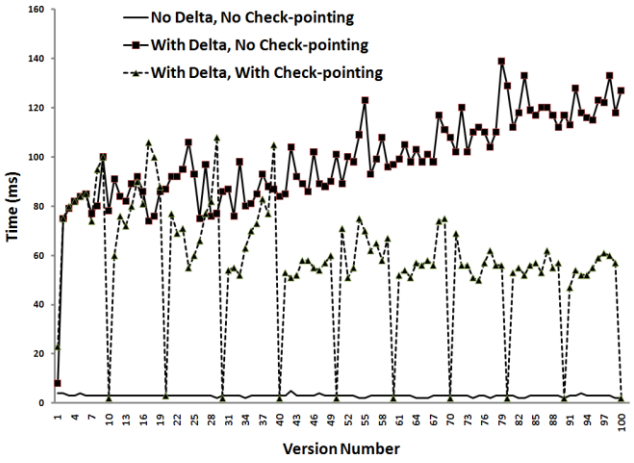


**Figure 8: File Write Time - M Workflow**

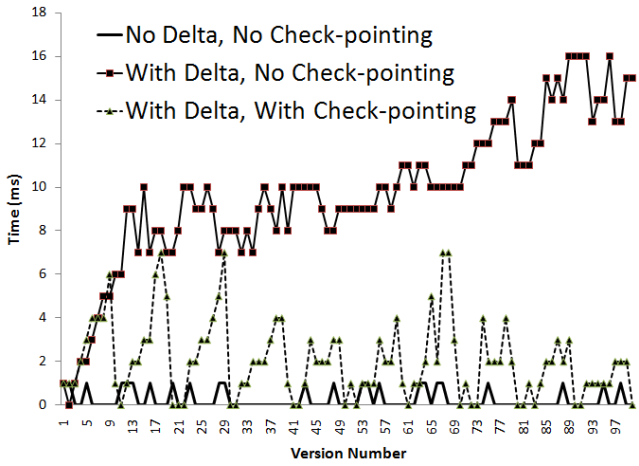


**Figure 9: Recovery Time - O Workflow**

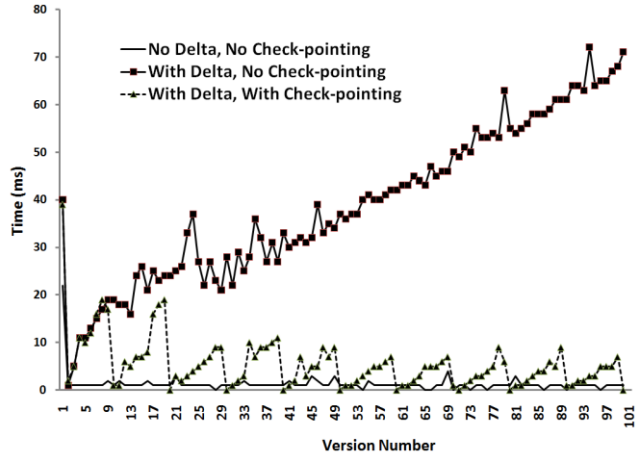


**Figure 10: Recovery Time - M Workflow**

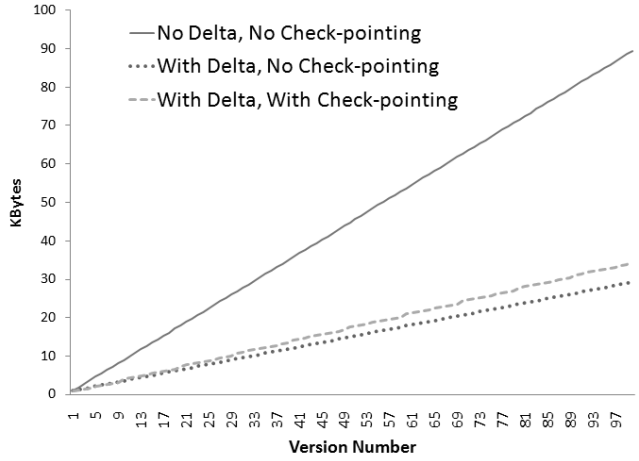


**Figure 11: Space Usage for a Version - O Workflow**

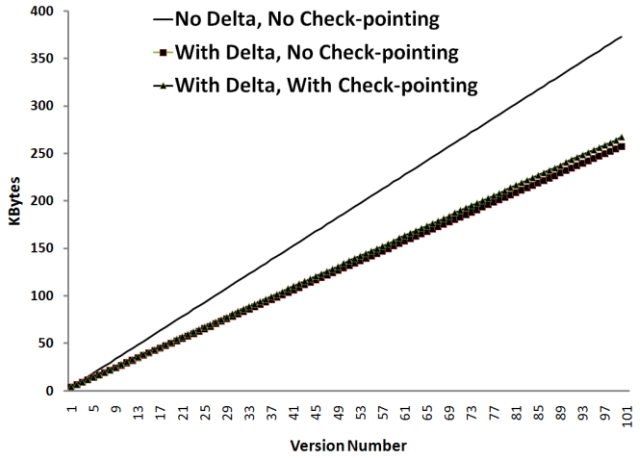


**Figure 12: Space Usage for a Version – M Workflow**

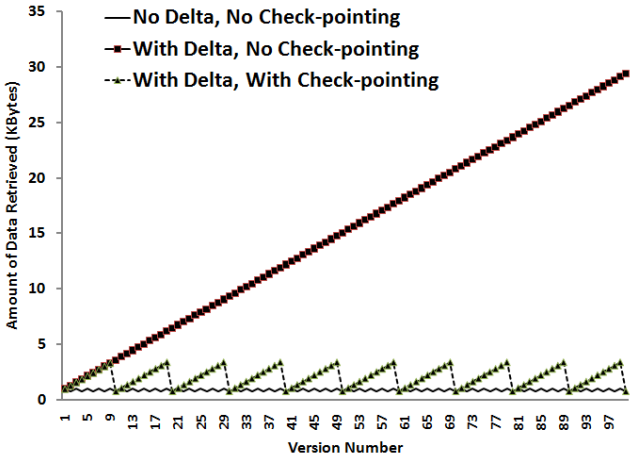


**Figure 13: Amount of Data Retrieved to Recover a Version - O Workflow**

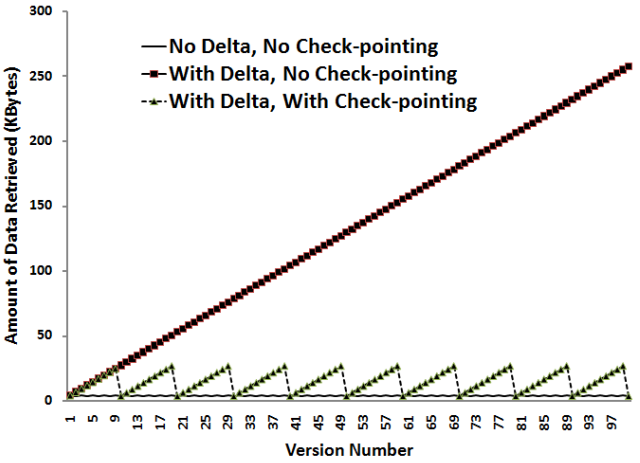


**Figure 14: Amount of Data Retrieved to Recover a Version – M Workflow**

Again for both the workflows, "No delta, No Checkpointing" option performs better, because to recover a version, it only needs to retrieve only that particular file from the file system. "With Delta, No Checkpointing" case is worst, because to recover a given version, it needs retrieve all the previous versions to re-construct the file. As proven in the graph, using "With Delta, No Checkpointing" option, the overhead to retrieve a given version is directly proportional to the version number of the file. The compromise case "With Delta, With Checkpointing", performs better than "With Delta, No Checkpointing" but not better than "No delta, No Checkpointing". For O workflow, "No delta, No Checkpointing" option's overhead is at most 3 times better than the other two options, whereas for the M workflow it is 15-20 times better.

## 7. Discussion and Future Work

We evaluate three options for maintaining the different versions of an object in the system. Though "No delta, No Checkpointing" options performs poorly with respect to storage usage (4-5 times for smaller workflow, smaller delta and 2-times for larger workflow, large delta) it outperforms both other options with respect to version save time (20-30 times for the large workflow, large delta and 5 times for smaller workflow, small delta) and version recovery time (10 times for the smaller workflow, small delta and (5 times larger workflow, large delta). So in selecting an option to maintain objects within the system, one should take following factors into consideration the size of the data objects, the average changes for data objects between different versions of the same object, and the response time to the user and the system.

By employing a strategy to dynamically adjust to the properties of different objects rather than adhere to a static policy, the system will perform better. The underlying registry implementation should be able to support saving new versions as deltas or as it is. If the deltas are used, recovering a given version can be achieved with or without checkpointing. Figure 2 captures the artifacts EVF can version in an experiment. Even though output data is versioned within the EVF framework, it can be a challenge to visualize them if the relevant software is not available later. Our implementation enables visualization of output data products stored as images. The system can be extended to integrate other visualizations tools within Trident enabling scientists to visualizations within Trident itself.

We are using the Trident Workflow Workbench in LEAD II, and expect to use EVF in practice in that context for exploring reproduceabiltity and experiment recreation. We are integrating real time data sources into Trident to be used within EVF, using the versioning control services provided by those services. An interesting exercise is to reproduce a workflow, retrieving previous version data inputs from those data sources, and also the previous versions of workflows within EVF framework to reproduce a previously run workflow.

## 8. References

[1] Amazon S3 Web Service. http://aws.amazon.com/s3

[2] Microsoft SQL Server Data Services (SSDS). www.microsoft.com/sql/dataservices/default.mspx.

[3] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. *In Proceedings of the 5th Symposium on*

*Operating Systems Design and Implementation (OSDI)*., 36(SI):1{14, 2002.

[4] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. *In Proceedings 16th International Conference on Scientific and Statistical Database Management, 2004*, pages 423-424, 2004.

[5] P. Andrew, J. Conard, and S. Woodgate. Presenting Windows Workflow Foundation. 2005.

[6] R. Barga and L. Digiampietri. Automatic generation of workflow provenance. *Lecture Notes in Computer Science*, 4145:1, 2006.

[7] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, and Y. Simmhan. The Trident Scientific Workflow Workbench. *In IEEE International Conference on eScience*, pages 317-318, 2008.

[8] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys (CSUR)*, 37(1):1-28, 2005.

[9] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data & Knowledge Engineering*, 24(3):211-238, 1998.

[10] R. Chatterjee, G. Arun, S. Agarwal, B. Speckhard, and R. Vasudevan. Using applications of data versioning in database application development. *In ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 315{325, Washington, DC, USA, 2004. IEEE Computer Society.

[11] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming scientific and distributed workflow with Triana services. *Concurrency and Computation*, 18(10):1021, 2006.

[12] E. Deelman, J. Blythe, Y. Gil, C. Kesselman,G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. *Lecture Notes in Computer Science*, 3165:11-20, 2004.

[13] K. Droegemeier, K. Brewster, M. Xue, D. Weber, D. Gannon, B. Plale, D. Reed, L. Ramakrishnan, J. Alameda, R. Wilhelmson, T. Baltzer, B. Domenico, D. Murray, A. Wlson, R. Clark, S. Yalda, S. Graves, R. Ramachandran, J. Rushing, E. Joseph, "Service-oriented environments for dynamically interacting with mesoscale weather", *Computing in Science and Engineering, IEEE Computer Society Press and American Institute of Physics*, 7(6):12-29, 2005

[14] M. Flouris and A. Bilas. Clotho: Transparent data versioning at the block I/O level. *In Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST 2004)*,pages 315-328, 2004.

[15] J. Freire, C. Silva, S. Callahan, E. Santos, C. Scheidegger, and H. Vo. Managing rapidly-evolving scientific workflows. *Lecture Notes in Computer Science*, 4145:10, 2006.

[16] J. Frew and R. Bose. Earth system science workbench: A data management infrastructure for earth science products. *In Proceedings of the 13th International Conference on Scientific and Statistical Database Management*, pages 180-189. IEEE Computer Society, 2001.

[17] C. Goble and D. De Roure. myExperiment: social networking for workflow-using e-scientists. *In Proceedings of the 2nd workshop on Workflows in support of large-scale science*, page 2. ACM, 2007.

[18] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic acids research*, 34(Web Server issue):W729, 2006.

[19] P. D. In, J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger. Self-securing storage:. *In Symposium on Operating Systems Design and Implementation*, pages 165-180. USENIX Association, 2000.

[20] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, et al. Oceanstore: An architecture for global-scale persistent storage. *ACM SIGARCH Computer Architecture News*, 28(5):190-201, 2000.

[21] E. Lee and R. A. Thekkath. Petal: Distributed virtual disks. *In Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84-92, 1996.

[22] J. P. MacDonald. File System Support for Delta Compression. Master's thesis, University of California at Berkeley, 2000.

[23] J. Michalakes, J. Dudhia, D. Gill, J. Klemp, and W. Skamarock. Design of a next-generation regional weather research and forecast model. *Towards Teracomputing*, pages 117{124, 1998.

[24] C. Pancerella, J. Hewson, W. Koegler, D. Leahy, M. Lee, L. Rahn, C. Yang, J. Myers, B. Didier, R. McCoy, et al. Metadata in the collaboratory for multi-scale chemical science. *In Proceedings of the 2003 International conference on Dublin Core and metadata applications: supporting communities of discourse and practice|metadata research & applications*, page 13. Dublin Core Metadata Initiative, 2003.

[25] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *In Proceedings of the thirteenth ACM symposium on Operating systems principles*, pages 1-15, New York, NY, USA, 1991. ACM.

[26] D. Santry, M. Feeley, N. Hutchinson, and A. Veitch. Elephant: The file system that never forgets. *In Workshop on Hot Topics in Operating Systems*, pages 2-7. IEEE Computer Society, 1999.

[27] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *ACM SIGMOD Record*, 34(3):36, 2005.

[28] M. Stonebraker. The design of the postgres storage system. *Morgan Kaufmann Publishers*. pages 289-300, 1987.

[29] I. T. Foster, J.-S. Vockler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. *In Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, pages 37-46, Washington, DC, USA, 2002. IEEE Computer Society.

[30] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, H. T. Vo. Vistrails: Enabling interactive multiple-view visualizations. *In IEEE Visualization, 2005. VIS 05*, pages 135-142