Reliable Management of Community Data Pipelines using Scientific Workflows[†]

Yogesh Simmhan, Roger Barga, Catharine van Ingen
Microsoft Research, Redmond
{yoges, barga, vaningen}@microsoft.com

Alex Szalay
Johns Hopkins University, Baltimore
szalay@jhu.edu

Jim Heasley
University of Hawai'i, Manoa
heasley@ifa.hawaii.edu

Abstract

The pervasive availability of scientific data from sensors and field observations is posing a challenge to data valets responsible for accumulating and managing them in data repositories. Science collaborations, big and small, are standing up repositories built on commodity clusters need to reliably ingest data constantly and ensure its availability to a wide user community. Workflows provide several benefits to model data-intensive science applications and many of these benefits can be transmitted effectively to manage the data ingest pipelines. But using workflows is not panacea in itself and data valets need to consider several issues when designing workflows that behave reliably on fault prone hardware while retaining the consistency of the scientific data, and when selecting workflow frameworks that support these requirements. In this paper, we propose workflow design models for reliable data ingest in a distributed environment and identify workflow framework features to support resilience. We illustrate these using the data ingest pipeline for the Pan-STARRS sky survey, one of the largest digital surveys that accumulates 100TB of data annually, where these concepts are applied.

1. Introduction

The ever increasing amount of scientific data from shared instruments, field observations and publications is posing a challenge to accumulating and managing them [1,2,3]. Science collaborations, big and small, are standing up shared repositories of community data administered by data managers ("data valets") who are responsible for loading, curating, cataloging, and publishing shared scientific datasets that grow large with time and support a wide user community. Many of these data repositories are sufficiently large that they must be distributed across commodity clusters; many of these also change frequently enough that they can no longer make use of traditional backup for fault recovery. Closely tracking the data ingest process is important when harmonizing diverse data from multiple sources. Lastly, several of the data management tasks call for domain expertise and knowledge of the data, causing scientists rather than system administrators to increasingly be involved.

Workflows have provided several benefits over traditional scripts for data-intensive sciences precipitating their adoption [4,5]. Scientific workflows have been used by scientists to compose, execute, and monitor e-Science applications and *in silico* experiments across desktops, high performance clusters and the Cloud. Workflows automate operations and reduce the time required to make the data available to the community. They provide provenance tracking and monitoring capabilities, and graphical tools for composition using standard and domain specific activity libraries. They also support heterogeneous execution environments and are well suited for data flow tasks. Many of these benefits can be carried over to the data ingest and management pipeline [6].

But using workflows is not panacea in itself and data valets need to consider several issues when *designing data ingest workflows* and *selecting workflows frameworks* to build a robust repository. Data ingest pipelines have a higher requirement for fault resilience as failures impact a large user community rather than only an individual scientist. Moreover, fault tolerance is not just the ability to run the same workflow again as workflows have side effects. For example, reloads may cause duplicate data which can trigger data validation errors and/or skew science results. This makes provenance collection to track the progress of the data updates through the pipeline paramount. Valets are not just responsible for making the data available to the user community in a timely fashion, but also for ensuring continued availability of the data to that community. Deploying a shared data repository on commodity hardware also creates a need for workflows to handle hardware faults. While older fault tolerance approaches to data management focused on stopping faults at the lowest levels in the system (e.g. through RAID), the contemporary model is to accept that there will be faults and minimize its impact on the user (e.g. through replication) [7,8,9]. The data valet workflow system must support distributed data movement, distributed recovery tasks, and coordination among workflows. The workflow framework must be able to handle a large number of workflows that may execute in near real time in response to new data arrivals where the ability to buffer those arrivals may be limited.

[†] *A short version of this paper is to appear in the IEEE eScience 2009 conference*

In this paper, we discuss some of the issues in designing and building reliable data loading pipelines using scientific workflows to support shared scientific data hosted on commodity clusters. We illustrate these using the data ingest pipeline for the Pan-STARRS sky survey, one of the largest digital surveys that is soon to go operational [10]. Pan-STARRS uses the Trident workflow workbench [11] to ingest and manage 100TB/year of data arriving continuously from the telescope onto SQL Server databases distributed across a commodity Windows HPC cluster [12,13]. Data ingest workflows in Pan-STARRS are designed with the awareness of running and managing data in a fault-prone, distributed environment using the models described in this paper. These help Pan-STARRS provide a far higher guarantee of data availability and drastically reduced time-to-science for hundreds of astronomers soon to use the system. We make two specific contributions in this paper:

1. We present storage systems features and data layout design to ensure reliable data access.
2. We propose workflow design models for reliable data ingest and management in a distributed environment
3. We identify workflow framework features required to support the workflow models and fault resilience

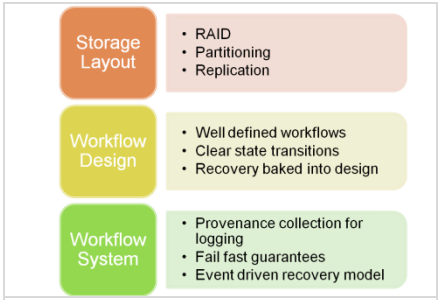

Figure 1. Summary of reliability techniques for data pipelines.

The rest of the paper is structured as follows: in Section 2, we provide a background of the Pan-STARRS sky survey project and its data ingest pipeline along with the Trident workflow workbench; in Section 3 we explore data level reliability, including storage reliability and data layout design, and their application to Pan-STARRS; in Section 4 we describe the workflow design models that help data valets compose reliable data workflows with examples from Pan-STARRS; in Section 5 we identify workflow system features that can support robust valet workflows for data repositories and highlight

it with the Trident workflow workbench used in Pan-STARRS; in Section 6, we present related work on workflow and data repository reliability; and we present our conclusions in Section 7.

2. *The Pan-STARRS Data Ingest Pipeline*

The Panoramic Sky Survey and Rapid Response System (Pan-STARRS) is a next generation telescope being constructed at Hawaii to detect near Earth objects that may collide with our planet and will perform a detailed survey of the solar system and nearby galaxies [14]. PS1 will generate 1PB of image data each year and reduce this to 100TB of astronomical detections through an Image Processing Pipeline (IPP). These 5 billion objects and 100 billion annual detections of them are continuously updated as the survey progresses and made available to the astronomy community to performinvestigations. Data and metadata arriving from the IPP are ingested into SQL Server databases in the Object Data Manager (ODM) layer. Astronomers are accustomed to using databases to store, query and transform their data [15], and in Pan-STARRS they use the CASJobs batch query portal interface to access the repository [16]. CASJobs rewrites the user queries to map logical database names to physical

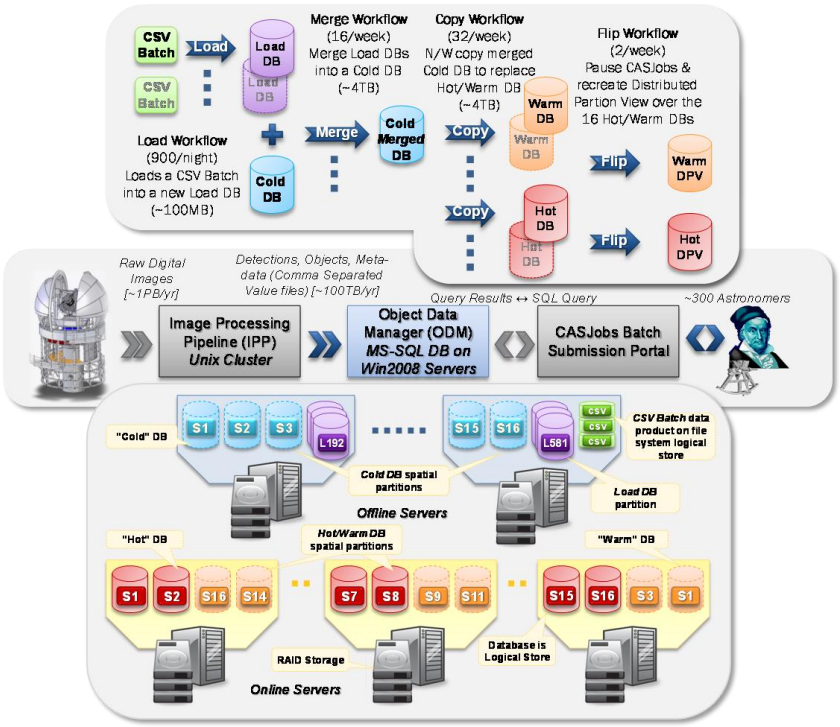

Figure 2. ***(Middle)*** The data flow between the Image Processing Pipeline (IPP), Object Data Model (ODM) and CASJobs. The focus of this paper is on the ODM. ***(Bottom)*** The databases in ODM are spatially partitoned across commodity hardware with multiple replicas. ***(Top)*** The Load, Merge, Copy and Flip workflows form the backbone of the data ingest pipeline.

ones, provides queues to submit the queries on, load balances the query execution, and write results to *MyDB* personal databases. About 300 astronomers are expected to actively use the data for studies. The Pan-STARRS approach is an evolution of Sloan Digital Sky Survey (SDSS) [17] improving upon it in two key respects: it collects 30 times more data, and makes it available to the community within 1 week of observation instead of 6 months.

Output from the IPP is in the form of FITS files which are converted to Comma Separated Value (CSV) files by a DX preprocessing step in the ODM (Figure 2, middle). *Detections* and *objects* are the two main batch types that arrive. Each photo frame captured by the telescope results in a batch of CSV files that form the unit of data ingest; about 1000 arrive each night, a majority of them being *detections*. The data valets of the ODM use scientific workflows based on the Trident scientific workflow workbench to build the pipeline that reliably ingests CSV batches into the SQL repository [6]. The *Load workflow* (Fig. 2, top) inserts the contents of a CSV batch into a new Load database or *Load DB* (wlog, a new schema within a database instance) where the data is validated and derived columns computed. Load workflows are the work horses and 6000 run each week.

The Load DB's act as staging databases that are merged with existing detection database contents at the end of each week; this balances between freshness of data available to astronomers and the cost/complexity of the merge. The detections databases form the bulk of the repository contents and are spatially partitioned across 16 database instances distributed on commodity servers (Fig. 2, bottom). This enables capacity scaling over time by migrating databases to additional servers and fault tolerance due to decentralized storage. There are Hot, Warm and Cold copies of the detections databases on different machines for robustness; the cost of realtime SQL replication is prohibitive for the data sizes involved. The *Hot* and *Warm DBs* are online running CASJobs user queries, while the *Cold DB* is used for offline data ingest. *Merge workflows* run each week to merge the 6000 incoming Load DB's with the 16 Cold DB replicas, one workflow per Cold DB. The merge workflow performs consistency checks of incoming and existing data, combines both data to create an updated copy, and rebuilds indices. Each merge performs 4TB of data rewrites. Subsequently, the *Copy workflow* copies the updated Cold detection databases (all 100TB) over the network to the machines with the Warm replica, followed by a *Flip workflow* that pauses user jobs on the online Warm and flips it with the newly copied and fresher Warm replica. The same is repeated for the Hot copy, ensuring that there is always one Hot/Warm copy available for user queries. A garbage collector workflow later deletes all Load DBs that were successfully loaded and surfaced to the users that week.

a. Trident Scientific Workflow Workbench

Trident is a scientific workflow workbench that leverages the capabilities of a commercial workflow engine, .NET Windows Workflow Foundation (WF) that ships as part of the Windows OS [22,11,6]. Trident uses WF as the core engine but adds features required by scientific workflows, resulting in a dependable and maintainable project. Trident workflows are composed out of activities that may be any .NET object that derives from a standard base class. Common activities that wrap a web service or SQL stored procedures are provided, and activities can be extended to meet needs to science domains. Workflows are visually composed using a graphical tool that supports both data flow and control flow semantics. Workflows and activities are registered with a Registry service that is backed by an XML file store, relational database or in the Cloud (SQL Server, Amazon S3 and SQL Azure). The registry acts as an authoritative catalog for all resources in the system, including workflows, activities, services, compute nodes, data products, and provenance, and supports private and shared user spaces.

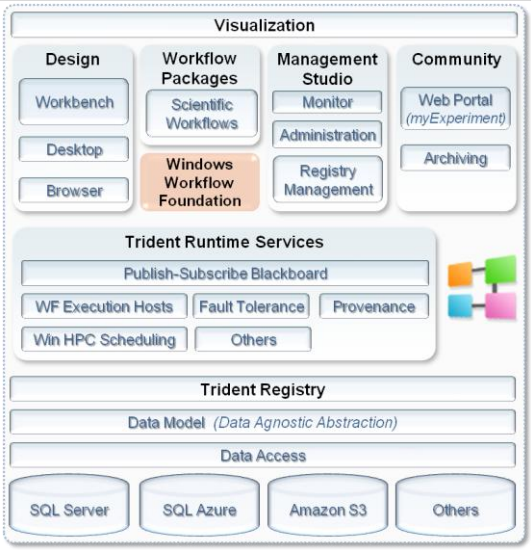

Figure 3. Logical Architecture of Trident Workbench

Workflows can be launched from the composer and are executed by the WF runtime either on the local desktop or on a remote Windows HPC cluster. The execution service is responsible for selecting the node to run the workflow based on resource, time or affinity policies, queuing the workflow instance, and creating the WF runtime with associated DLL libraries required by the workflow. Users can track the workflow progress in the composer, on the desktop or online through a SilverLight plugin, through events published by the workflow runtime using the BlackBoard publish-subscribe system [50]. These provide both resource usage monitoring and provenance tracking capability. Workflows, their provenance, and generated data products can be shared between users of the Trident registry, or online with the community though the

myExperiment portal [51]. The Trident registry is also extensible to work with external catalogs and the composer interface can be integrated with generic or domain specific visualization tools such as COVE.

3. Reliable Data Storage and Layout

When building a growing community data repository, it is useful to build layers of abstraction in data storage to ease manageability and design for reliability within each layer. One convenient and common way of layering is to separate the physical storage medium from the logical stores. For e.g., the physical storage may be a set of hard drives (JBOD) on one or more servers, while the logical drives could be drive letters or mounts, with multiples logical drives in one physical drive, or logical drives that span multiple physical drive. The data itself is laid out on the logical drives using different layout schemes. These are discussed below.

a. Partitioning

Partitioning datasets helps scope the data and data access loss when faults occur [52,53]. Partitioning provides a graceful degradation of service where data (access) loss is limited to only those partitions that reside on the same logical store that fails while the other partitions are available for access or query [48]. The two key aspects of partitioning data are the *partition function* used to split data into partitions and the *placement* of these partitions. Partition functions group the data into partitions based on a partition key that is a property of the data, and the partitions themselves are placed on logical stores. Partitions form the unit of data management, and logical stores the unit of data storage.

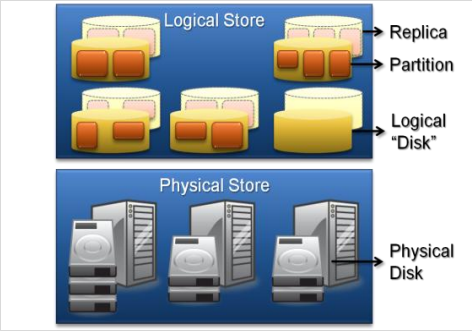

Figure 4. Physical & Logical data storage

Partition Function: The data sets may either be globally partitioned on a particular key they all share (e.g. lat/long coordinates in geospatial data), or they may be grouped by the types of the data (e.g. all MODIS data does to a particular partition) or other domain specific attributes. The hash or partition function maps this partition key to the partition in which the data resides.

The partition function used affects the usefulness of the available data in case of faults, so care must be taken to group related datasets required to perform science tasks together. For example, it may be necessary to have the metadata/header file for a data file for the data file to be used. So ensuring that the metadata/header file is collocated on the same partition as the data file is important, else loss of the logical disks holding the metadata partition can make the data file unusable even though the file itself is accessible. The partition function must be easy and cheap to compute since it is computed often, both when writing and reading the data. If the function is numeric/simple and global to all datasets, it may be used as a canonical function even within the code. Often the function takes the form of an entry in a registry that acts as a lookup table for performing the mapping.

The number of partitions of data is dependent on the partition function and as is usual in a hash function, it is important to ensure that the keys are uniformly distributed across partitions. However, since the partitions map to logical stores, it is useful to have a hash that keeps the partition size (i.e. the total size of the data stored in each partition) uniform so that all partitions can be treated equally. One factor to consider when determining this is the rate of growth of data. Just as all partition keys may not map to similar data sizes, not all data grow equally in size with time. For e.g., a collaboration may undertake specific campaigns at times that cause a large influx of data of a certain type. Having the ability to handle this, either within the partition function or by changing the partition function, would be useful. The partition function may also need to change the partition sizes grow beyond the size of the logical store with time. If the change in partition function causes a drastic remapping of the data, the repartitioning may need costly movements between logical stores. So a suitable partition function that can be expanded over time should be chosen.

Partition Placement: The process of selecting the logical drive to store a partition in part of the data layout process. While the logical abstraction over the store and partition abstraction over the data allows arbitrary mapping from partition to store, data layout since it can affect the usability of data in the system. Data valets need to consider several factors when making a data layout decision. Keeping partitions containing data that are often accessed together on the same logical store improves data locality and helps perform efficient data joins and computation locally if possible. This can also help with data consistency: when a logical store fails, it takes makes both the related partitions unavailable. On the other hand, having different partitions on different logical stores can potentially improve the data access speed by performing parallel I/O access on the logical stores (assuming they are on independent physical stores) – I/O access if often the bottleneck in data intensive applications. It is important to

study the data access and query/computation patterns to evaluate the trade off when determining the partition placement.

Applications accessing the data should also be aware that different partitions may be on different logical stores, and that some of the logical stores may have failed due to faults. This awareness can help application to operate on partial data and provide partial results, rather than failing completely. For e.g., when performing a weather forecast simulation that always uses land cover data and precipitation data together, one layout approach is to co-locate the two data types on the same logical store (or even partition) but have different spatial regions to be on different logical stores. This way, if we lose a logical drive, we will not be able to perform a forecast on the spatial region that on that store but can successfully run the simulation on other regions since both land cover and precipitation data are available. The forecast application may even be aware that these regions are on different logical disks and perform parallel I/O across these stores.

b. Replication

Replication of data ensures redundancy and availability of data [40]. Just as a RAID layout provides backup copies at the physical storage layer, replication can be done either at the logical store level (e.g. logical disk volumes, databases) or the partition level (e.g. files). Replicas are copies of the replicated data and the replicas may strongly consistent, where in all the replicas always have the same contents, or weakly consistent, where there may be a period of time when the replicas have different values but they will eventually become the same. Some distributed file systems [8] and databases [54] provide strong replication while a regular file backup or synchronization service is an example of weak replication [55].

Data replication can help improve the reliability by having an alternative copy of the data in case an existing copy is lost. More the number of replicas, higher the reliability but the trade off is a higher cost of maintaining the replicas, increased storage and potentially greater complexity of replica recovery – unlike physical data mirroring that is transparent to the user and application, data valets may need to be aware of the replication technology used and incorporate replication/recovery features into their application/workflows. Replication can also help improve the performance by having multiple copies of the data to access.

c. Storage and Data Layout in Pan-STARRS

Pan-STARRS uses commodity clusters to host its data in the ODM layer using the GrayWulf architecture [12]. The servers in the cluster are dual quad-core machines with 16GB memory running Windows 2008 and form a Windows HPC cluster. There are multiple configurations for the different server roles with the data server having between 20-28 750GB SATA disks. The physical storage is a RAID10 [56] stripped and mirrored setup to provide failover. Dual RAID controller allow maximal utilization of parallel I/O across the disks. RAID mirroring provides transparent data replication at the physical level that allows recovery with minimal effort when one of the mirrored disks fails.

Most data in Pan-STARRS is present in databases and we treat databases as the logical storage level. Due to the large sizes of the data being loaded and the high cost of transactional updates, the logging features of the databases are disabled for updates to the databases. This means that many of the ACID features of databases cannot be relied upon and have to be handled by the applications performing writes to the database, effectively treating them as files. The advantage of databases for Pan-STARRS is in the ability of users to perform efficient queries over the indexed data.

Detections and objects are maintained in separate databases, one reason being the detections change more frequently with time than the objects. Since the data is spatial in nature, they are spatially partitioned using the declination and right ascension for each object and detection (analogous to latitude and longitude for geospatial data) using a spatial partition function [57] that maps them to a unique integer. Since the objects in the sky are not uniformly distributed, more detections will arrive from some regions than other. So the integer hash for the *RA* and *dec* is divided into ranges such that there are approximately equal numbers of objects within each range, and these ranges are stored in a lookup table. So the partition function is a combination of a mapping function and a lookup, and each range corresponds to one logical partition for storage.

4. Reliable Workflow Design

Ensuring reliability in any distributed environment is non-trivial. The workflow design can have a big impact on the reliability guarantees that are possible when ingesting and maintaining community data on commodity clusters that are prone to failures. A good design makes it possible to analyze all fault paths, hardware and software, and define the corresponding recovery paths in a modular manner. Some of the effective design practices we have identified are discussed below.

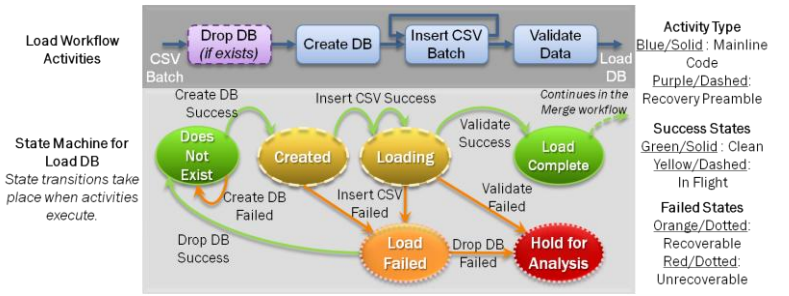

Figure 5. *Load Workflow* and state transition diagram for Load DB it operates on.

a. Well-defined, well tested workflows

Unlike the exploratory workflows composed by scientists, valet workflows are run repeatedly with infrequent modifications to the workflows and activities [4]. Changes to workflows or activities happen rarely, for example when a new data product is incorporated or if the data layout is changed. New or changed workflows and activities need to fit with the operating schedule of the repository and transitioned to operations judiciously to prevent jeopardizing the ongoing operations. Moreover, the impacts of valet workflows on the repository contents tend to be cumulative – data errors due to software bugs can easily accumulate to corrupt the entire repository. The combination of high frequency of operational use and the implications for error means that all valet workflow activities and any changes to them must be thoroughly tested.

The *Load workflow* in Pan-STARRS is an example of such a work horse workflow that is run about 900 times per day every day in the year to load the CSV batches arriving from the telescope. A small bug or leak in the implementation can be magnified quickly. The consequence of an ill formed activity can be disastrous and affect a wide user community [6].

The burden of verification of workflows and activities is reduced when there are relatively few unique workflows and activities and their semantics are crisp and clearly specified. It can be hard to analyze large workflows with complex execution paths and even harder to recover them. All activities and workflows should expect to fail and the *known* exceptions they "throw" documented and, preferably, exposed as part of their interface.

b. Granular, reusable workflows

Workflows developed by data valets also differ from their scientist counterparts in the non-exploratory, precise nature of the tasks they perform. Valet activities and workflows should be developed in a reusable manner rather than custom built. Activities and sub-workflows should be used as building blocks and workflows which encapsulate policy should be separated from workflows which are purely mechanistic. Such workflows are then well suited to modular composition through hierarchical nesting or sequential chaining of activities and sub-workflows. Activities should also be parameterized and use indirect references where possible, both to enable reuse and avoid locality and file-path assumptions given the distributed nature of the system.

The *Copy* and the *Flip workflows* in Pan-STARRS are usually executed sequentially and could have been written as a single *"Copy & Flip" workflow*. However, designing them separately allows the Copy workflow to be reused individually: to copy the weekly merged Cold DB, to recover a failed Hot database from the Cold replica, or for any other generic data movement purpose. The Copy workflow also has an associated "preamble" workflow. That preamble determines the destination of the copy and includes logic for load balancing and fault recovery. Only the preamble workflow has knowledge of whether the Copy workflow is triggered by normal data publication, fault recovery, or for generic data movement. Note also that the testing of the preamble can be independent of the Copy.

Within the Copy workflow are a number of reusable activities for parallel file copy. These activities are built as a suite of well tested file management activities for common tasks like network copy, parsing file formats and extracting metadata.

c. Workflows as State Machines

The state of the data repository is the union of the states of its constituent data products. It helps to consider workflows as acting on a state machine, taking the data products from one state to another. The data valets need to know the status of the repository at any point in time, to see if the pipeline is healthy and all data products are being surfaced to the users. Modeling the results of the workflows as nodes of a state machine enables simple discovery of that status. The nodes of this state machine refer to the status of the data being manipulated and the transition edges are workflows or activities that operate on the data to take them to one of the subsequent states [18,19]. Tracking the states of the data products acts a simple form of *data* checkpointing [20] and attempt to provide the workflows with a certain degree of transactional capability.

These are a couple of advantages to this approach:

- Valets often find it more meaningful to determine the state of the repository in terms of the health of the data products rather than the progress of workflows. This is particularly the case when there are many workflows in progress and the data products are operated upon by multiple workflows over different points in time. For example, consider the difference between the Pan-STARRS' 16 weekly *Merge workflows* and the 6000 *Load*

workflows occurring during the week. The failure of a Load workflow has a low impact: it just affects the freshness of data for a single photo frame, does not impact ongoing science queries, and should be easily recoverable. In contrast, the loss of a Merge workflow will affect a science user as the Warm DB goes offline while the associated Cold detection database is recovered from it; all scientist queries executing on that Warm DB are impacted. From the view of the valet, knowing that the Cold and Warm databases are currently unavailable for data processing workflows or user queries is more useful than the failure of the merge workflow.

- Data valets need to identify the potential fault states and the workflows/activities that can cause data to arrive at that state, and define recovery paths from that state. Valets use workflows as a means to the end of successful data repository operations – to get data from incoming to ingested state, stale to fresh, and missing to available. This is a goal driven approach to designing workflows at an abstract level. Given a data product type in a given state, it allows them to determine the workflow(s) that will take it to a target state.

Data states can broadly be classified into their initial state (clean), intermediate states when they are in-flight, final state (clean) when they are usable, and fault states which they can be recovered from or not. Since activities cause transition from one state to another, they should be of sufficient granularity to cause at most one state transition on one or more data products. An activity is a blackbox and should not perform complex operations that leave the system in an unambiguous state. This allows recovery from a data state to be done in a deterministic manner since the data states that a failing activity can affect are known. This design also implies that consecutive activities that do not cause a state transition may be collapsed up into one for clarity. While there are workflow systems that orchestrate workflows as a finite state machine [21,22], it is not essential to use such a system as long as users design workflows with a data state model in mind. In fact, the model of a single activity causing a state change to multiple data products may not fit the state models of these workflow systems.

Figure 5 illustrates this approach using the state model for the *Load DB* in Pan-STARRS as it is acted upon by activities in the *Load workflow*. The Load DB goes from a `'Does not Exist'` state before it is created to a `'Load Complete'` state when the workflow executes successfully. The successful execution of the *Create DB* activity creates the first physical presence of the Load DB and takes it to the `'Created'` state. The `'Loading'` state is an intermediate step along the path of successful execution as each CSV File in the batch is inserted into the Load DB tables. If the insert fails, it transitions to the `'Load Failed'` fault state which is recoverable from – recovery of the Load workflow is performed at the granularity of the entire Load DB by executing the *Drop DB* "preamble" activity that does cleanup before re-executing the workflow. Data validation failures are serious and cannot be recovered automatically; the `'Hold for Analysis'` state allows manual intervention. When validation succeeds, the Load DB reaches the `'Load Complete'` state where the *Merge workflow* can use it for the end of week merge and cause subsequent state changes (not shown).

At any point in time, data valets can view the current state of any of the 30,000 Load DBs that are processed during the year to verify that they were loaded successfully into their target detection databases, even after the actual physical Load DB has been garbage collected after a successful merge.

d. Recovery baked into design

Faults are a fact of life for data ingest workflows. When recovery is baked into the workflow design, it ensures that routine operational workflows and/or activities are actually the same workflows that run during failures. This reduces both the software test burden as well as giving the operations staff experience in handling "normal" faults [13]. The faults in the workflow can occur due to failure of the machine running the workflow, loss of access to data or service on a remote machine, or even faults in the contents of the data. While each potential fault could be handled with a specific fine-grain targeted recovery, our design approaches are both coarser and, as a result, simpler.

Re-execute Idempotent Workflows: Whenever possible, a *workflow* should be idempotent [9,23]. In the event of a failure, the workflow can be re-run completely either on exactly the same target or a substitute, duplicate target. This may be possible when the data is not being modified in place and the source data is available in the valid state required by the workflow/activity. Idempotent workflows must also not undertake a lost cause of resuming previously failed workflow attempts; the workflow should be of short duration. The workflow simply needs to execute to completion or fail.

The *Flip workflow* in Pan-STARRS is an example of this. The Flip is a short workflow that coordinates with CASJobs and executes a SQL query to rebuild the distributed partition view across the 16 Hot/Warm detection databases that have been updated at the end of the week. In the event of a failure, the entire flip is re-executed without any side-effects.

Resume Idempotent Workflows: Whenever possible, *activities within a workflow* should be designed to be idempotent, whereby a re-execution of the previously faulting workflow with the same input parameters ensures that

the data recovers [9]. The choice of whether to restart the entire workflow or resume from a given state depends on the characteristics of the data, the cost of not resuming the workflow, and the complexity of recovery involved. It may be possible to resume a workflow at the idempotent activity at which the fault occurred provided all preceding activities are also idempotent. In such cases, the workflow needs logic at the beginning allowing it to "jump" to an intermediate activity and resume from there.

The *Copy workflow* in Pan-STARRS is a resume idempotent workflow that performs parallel copy of database files over the network. Given that a total of between 2 to 4 TB of data are moved in each Copy, it makes sense to recover from a fault by skipping the files that have already been moved successfully and jumping to the iteration at which the failure took place and overwriting any partially copied file(s).

Recover-Resume Idempotent Workflows: Other workflows may be reduced to an idempotent model after the necessary cleanup to "rollback" the fault. A set of cleanup activities will check the current state of the data and take corrective steps to bring it to a consistent state before re-executing or resuming the workflow. This recovery can be "baked in" the workflow design by the valet designing that workflow, who has the best knowledge on how it can be locally recovered. There are two different cases of recovery: active recovery and preamble correction.

- ***Preamble*** correction, or ***passive recovery*** policy can be used in cases where interactions between workflows and a machine fails causing collateral damage, or when a simple cleanup will suffice. A Pan-STARRS example is the transient loss of network during a load. Re-executing the *Load workflow* will cause the recovery preamble ***activity*** to drop the existing Load DB if it is in an "unclean" state (Figure 5) and re-execute the mainline Load workflow activities.
- ***Active recovery*** is used when a separate recovery workflow is necessary for a particular fault or multiple faults. The failed workflow cannot proceed until a specific set of activities have been performed. The Pan-STARRS example is failure of a *Merge workflow* due to a machine crash. The long running merge consists of a number of idempotent update activities. In the event of failure of an activity, there is an explicit cleanup operation necessary to repair the underlying database files prior to resumption of the merge. That cleanup is achieved by a recovery workflow and then the previously executing merge can be resumed.

Key to this approach is that workflows and activities are designed to fail fast [24]. While it is tempting to write activities that are "fault tolerant", the activity may not have sufficient context to perform local recovery and its attempts could even exacerbate the problem. Bubbling up the fault allows recovery to be done at the point when a workflow fault could be "caught" and recovered (i.e. like a `try{}catch{}` statement) or handled as a preamble at the beginning of the workflow. While a catch presumes that the workflow instance is still alive to perform the cleanup after it fails, this is misplaced when the machine running the workflow crashes or the workflow instance process is forcibly terminated. An active recovery model can handle cases where the catch fails, out of memory exception, machine crashes, and other hard-faults, but requires the extra effort of writing the recovery workflow. Both the active recovery and preamble correction treat soft faults as hard faults and both are run just in time.

Recovery Workflows: Where possible, we reduce any global recovery problem to one of the earlier workflow recovery models for local recovery. Re-execute and resume workflows work for cases where workflows run independently of each other and can be recovered locally. When complex interactions between workflows take place to transform data, a global resource data synchronization may be required before meaningful recovery can take place. Fault recovery workflows can be designed to perform sanity checks of the states of the affected data products and bring them to a consistent state. There may also be performance and availability considerations in scheduling the recovery workflows in a certain order and at a certain priority. For complex scenarios and during the early phase of systems operations, these workflows have to be manually launched by the valets after investigating the states of the different data products for the simple reason that the valets need to watch the subsequent execution.

An example in Pan-STARRS is a machine failure causing the loss of a recently merged Cold DB and any accumulated Load DBs. First and foremost, the Cold DB has to be recovered to its pre-merge status from the associated Warm DB to avoid catastrophic data loss (loss of all 3 replicas). This requires coordination with the CASJobs scheduler running user queries on the Warm DB to take the scientist query queue offline, preempting or waiting for any Copy workflows that are accessing the machine on which that Warm DB resides to avoid disk and network contention, and then launching a new Copy workflow to restore the Cold from the Warm replica. After restoration, any accumulated CSV Batch loads must be re-executed. This ensures that the next scheduled Merge workflow will merge not only any new batches accumulated during the outage but also all batches that were previously merged, but undone as a result of the restore. While automating all of that is theoretically possible, most valets are going to want to monitor that.

5. Reliability Features in Workflow Systems

Using a scientific workflow system to design, build, and operate a reliable data ingest pipeline places specific requirements on the capabilities of the workflow system. Some features, such as provenance logging, become more important and must be inherently reliable. Others, such as simplicity of *ad hoc* workflow authoring are less important. This section discusses three necessary features and their attributes of a workflow system when used by data valets.

a. Provenance Collection for Logging

The ability to track and link provenance all the way from the source (telescope) to the community repository (Load/Merge workflow, Hot/Warm DB) to the results of analysis (queries) that the users perform is necessary to support verifiable science [25,26]. Tracking provenance of workflows that model scientific experiments is a key feature for scientists and many scientific workflow systems support provenance collection [27]. These tasks may happen across different workflows, users, environments and data products. Provenance systems usually record the execution of activity and workflows, and their inputs and outputs which include data products. While workflow provenance is often coarser and semantic in nature to support scientific exploration [28,29], valets need lower level system provenance [30,31] for a different purpose. A data ingest pipeline also presents scalability and reliability challenges for provenance logging.

Provenance for Data State Tracking. When workflows and activities model state transition between data, provenance provides a record of the transitions that take place. Provenance can be mined [32] to provide the current state of data products that is used in the workflow and recovery design. The actual state associated with the data product can either be *directly* inferred from provenance or *indirectly* derived depending on how the activities are designed.

- If activities expose the current state of the data product as input and output parameters, provenance can be queried directly to find the last activity that used/produced a data and the state for that data set in its output parameter. This requires the activity author to ensure the activity takes one or more `<data, state>` tuples as input and, if the state of the data is changed or a new data is created, generate their `<data, state>` tuples as output. This may include a fault output state. This has the advantage that the state can be easily established by a simple query for activity output values and is supported by most workflow provenance systems. The problem, however, is that a generic, reusable activity has limited information to determine the actual state of the data in the context of a larger workflow(s). For e.g., even though the *Insert CSV Batch* activity in Figure 5 changes the Load DB, it does not actually change the state of the Load DB beyond the first iteration. The author of the reusable *Insert CSV Batch* activity will not know this when developing it. Some workflow systems provide a work around by statically setting the state transition of an activity as a property of the activity composing the workflow [22].
- The alternative approach that is more generally applicable is to specify an external mapping between one or more (successful/failed) activity executions in a workflow to the state transition that they cause. This gives the flexibility to specify the state transition as a function of provenance without modifying the activities and can even handle interactions across workflows that cause a state to change. The challenge, however, is the need to specify such a mapping for all data product states which becomes difficult as the number of workflows and data states increase.

Pan-STARRS uses the latter approach of mapping. The provenance collected by the Trident workflow workbench is stored in a registry database in the Open Provenance Model [49]. Database views on the provenance tables capture the mapping between activity/workflow execution and the state of the data it operates on. These views are created for each type of data product, such as Load DB, Cold DB, etc. The state for, say, a particular Load DB would depend on the set of activities in the Load and Merge workflows that have executed on it (successfully or not), available as provenance for these workflows.

Provenance for Forensics. Provenance can also help investigate the exact cause of a fault beyond just reflecting the state of the data. Higher level provenance about the activity execution state and its input and output can help decide where the workflow failed and take immediate corrective action. Lower level provenance, akin to logging and monitoring systems, can provide information to track sources of transient errors or performance bottlenecks [30,31]. Such system level provenance can include disks and services used by an activity, CPU and I/O utilization, and concurrent activities and workflows running on the same machine. Since many workflows may be executing across the distributed system simultaneously, the ability to correlate the interplay among these can help debug transient errors. Provenance can also help discover error patterns or underperformance that may occur repeatedly due to faulty hardware or a buggy activity. Mining provenance for statistics about failures can help determine the cause. For e.g., a failing Copy workflow may be resumed from the point at which it failed using data

state derived from higher level provenance, but the mining lower level provenance may identify the cause as a disk going bad and causing repeated faults on workflows using that machine.

Scalable Provenance Collection. Workflow execution time varies from a few tens of seconds to many hours. The provenance collection overhead should not be a significant cost. The combination of granular workflows and constant data pipeline operation can imply a large number of workflow executions which in turn can generate a large number of provenance entries. Provenance must be retained for at least the time scale of a data release and potentially the lifetime of an instrument or collaboration. Moreover, querying provenance can be much more common in valet workflows because the provenance log is used as the operations log. Prior to the execution of a new workflow, provenance is often queried to determine the current state of all repository resources and hence the correct input/output parameters of that workflow. In Pan-STARRS, the provenance collected during the year will be used to annually determine if the 30,000 CSV batches generated by the IPP were successfully processed prior to creating a snapshot of the databases to be held as a static version.

Resilient Provenance Collection and Storage. On a scientist desktop, a lost provenance entry may create some confusion, but that confusion can possibly be rectified by the scientist. In a data repository pipeline, a lost provenance log entry means that the knowledge of the repository status is in doubt. The actual state – such as exactly which batches have been loaded – must be rebuilt by detailed forensic survey of all resources. Losing the entire provenance log means that forensic survey can be extremely onerous as it must begin at the last annual database snapshot. This creates a requirement for both reliable distributed provenance collection as well as replicating provenance in real time to avoid machine failures. For the Pan-STARRS system, the size of provenance collected is small enough to scale with real time SQL log replication.

b. Support for Workflow Recovery Models

Workflow frameworks may natively support some or all of the recovery model designs discussed previously. The ability to redo a failed workflow with the same input parameters and the same or different machine (in case of machine crash) helps rerun and recover *idempotent* workflows. Workflow frameworks may allow specific exception types to be associated with recovery activity paths so that faults generated by the workflow instance can be handled within the instance itself instead of having the user or an external entity initiate the rerun. These recovery paths may also be used to run a cleanup routine before terminating/restarting [22] that supports r*ecover & resume* workflows.

For *fault handling workflows*, the framework should provide the means to transmit the occurrence of the error with sufficient context to determine its cause and initiate recovery. This information may be provided by the provenance framework and/or by publish-subscribe systems, as in Trident, to notify the occurrence of a fault to an entity responsible for detecting and correcting it. The event or provenance state change can act as an automatic trigger to launch the recovery or may be logged for a user to interpret and determine appropriate action. The notification system itself has to be fault tolerant and provide guarantees of reliable message delivery [33]. Many scientific workflow systems support these features and the workflow recovery models we have presented should be portable to them.

c. Fail fast guarantees

Besides workflows and activities being designed to fail fast [24], the workflow framework itself should provide such an assurance when the failure is outside the control or knowledge of the activity/workflow. Unexpected errors in the workflow/software stack or error in the hardware – transient or permanent – need to be detected and reported with minimal delay. The workflow framework should provide protocols for detecting timeouts, liveliness of services, and to handle loss of network connectivity between the source of a fault and the fault handling entity. This allows an external observer to detect the fault early and deterministically to take corrective action.

The Trident framework is configured to be fail fast for Pan-STARRS. The Trident registry database is the central store for recording runtime, log and provenance information for workflows in the system. Any loss of connectivity to the registry causes workflows on the compute nodes to fail fast. While provenance can be collected asynchronously in Trident using a publish-subscribe system, the reliability requirements for Pan-STARRS are met by synchronously logging provenance with the registry database at the start and end of activity execution so that state is persisted for each activity before the next is started. Loss of connection to the registry causes the workflow to abort ensuring that when recovery happens, the exact progress of the workflow and the state of data operated by the workflow is known.

Trident also plugs into events generated by the Windows HPC scheduler through which workflows are launched onto the cluster where data reside. Any fault in the HPC fabric – job failure, machine loss, network outage, etc. – is communicated back to the Trident workflow service, through polling or callbacks, which are recorded as fault types in the registry. The registry database acts as a central point of reference for valets and their workflows, and is replicated in real time on a slave machine.

6. Related Work

There have been numerous projects that manage scientific data repositories on data grids and collaborations but their primary focus tends to be metadata management, long term preservation and building science portals for data access [34,35,36]. Reliability of repositories is often reduced to data reliability, usually handled by a replicated file system [1] or specialized hardware [37], and application reliability based on checkpointing and transaction semantics [38]. Our experience, reflected in this paper, shows that such a separation is not easy in operational data repositories that work with large data on commodity systems. Valet workflow and data repository reliability are intertwined. Even a temporary loss of data has a consequence on the ingest workflows and a fault in the workflow has side effects that can leave the data inconsistent and unusable.

Data replication for reliability is common in data grids and Cloud data centers [8,39]. Typically, replication depends on a replicated file system or database replication [40]. These handle data loss transparently by providing indirect addressing to application using the data and performing real time replica propagation. The goal is to mask failures from the application. We take a different approach of letting the workflows be aware of the replicated nature of data and bake in data recovery as a part of the workflow. This has two advantages – (a) the large size and frequent updates of the data has performance penalties on replicated file systems and databases. By replicating the data as part of the data ingest pipeline, we ensure that data is moved only when it is in a consistent state to be surfaced. (b) Applications using a replica that fails have to be recovered and there may be a sharp degradation in service as the lost replica is recovered. By making workflows aware that replicas can fail and actually having them perform the data recovery, we ensure that the recovery process is well coordinated and workflows resume in a deterministic manner once data is recovered. This is possible because we do not have to support general purpose applications but have full control over how the valet workflows are designed.

Scientific workflows have been dealing with data intensive workloads. The common model for frameworks to recovery workflows is to overprovision workflows [41] or monitor them for faults and re-execute/resume using provenance logs or checkpoints [42, 43]. Over-provisioning workflows is not possible in data repositories because of the shared nature of data. Multiple workflows operating on different copies of the data is a recipe for consistency nightmare if multiple faults occur. Checkpoint-restart assumes idempotent workflows and complete state capture. This is possible only if workflows are designed that way. Our workflow design models present further options robust design and the data state transition model help capture the state of the data, which is more important for recovery than the state of the workflow. We too drive recovery using provenance but allow mapping from provenance logs to data state views.

Transactional workflows [44, 45] in the business community have examined stateful processes and our designs are similar to some of these models. However, they do not deal with stateful data or data intensive tasks. Our work combines the features of both scientific and business workflow models [46] and applies these successfully to a real world scientific data repository for operational use. Much of our work builds upon well known concepts in distributed systems, fault tolerant applications and replicated databases: fail fast, statefulness, graceful degradation, idempotent tasks, asynchronous recovery, and eventual consistency [7,9,24,47,48].

7. Conclusion

The growing rate of scientific data requires reliable and generalizable models to build scientific data repositories that are continuously updated and shared with a broad user community. In this paper, we have presented a model for building a data repository on commodity hardware using scientific workflows to ingest data into the repository. We present different design principles for composing workflows that are recoverable, and maintain data in a consistent state by operating upon them as a state machine. This allows valets to compose reliable, goal driven workflows and provide them with the ability to track the health of their repository at any time.

The techniques we describe are agnostic to the scientific workflow framework; we identify key features that make the reliability guarantees possible and reduce the effort for the valets. Trident supports these and so do many other workflow systems. We have illustrated our work with examples from workflows operating in the Pan-STARRS sky survey where these techniques are used to manage 100TB of annual data to support 300 users.

Acknowledgement

The authors would like to recognize the contributions of the Pan-STARRS ODM team: Ani Thacker, Maria Nieto-Santisteban, Laszlo Dobos, Nolan Li, Michael Shipway, Sue Werner, and Richard Wilton from Johns Hopkins University, and Conrad Holmberg from University of Hawai'i for designing and deploying the data layout and

workflows. We would also like to acknowledge the efforts of the Trident team – Dean Guo, Nelson Araujo, and Jared Jackson, for building and shipping version 1.0 of the Trident Scientific Workflow Workbench.

Reference

[1] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, B. Moe, "Wide Area Data Replication for Scientific Collaborations," Grid, pp.1-8, 6th IEEE/ACM International Workshop on Grid Computing (GRID'05), 2005
[2] D. Agarwal, M. Humphrey, and N. Beekwilder, "A Data Centered Collaboration Portal to Support Global Carbon-Flux Analysis," Microsoft e-Science Workshop, 2008
[3] Beth Plale, Jay Alameda, Bob Wilhelmson, Dennis Gannon, Shawn Hampton, Al Rossi, and Kelvin Droegemeier, "Active Management of Scientific Data", IEEE Internet Computing, pp. 27-34, 2005.
[4] David Woollard, Nenad Medvidovic, Yolanda Gil, Chris A. Mattmann, "Scientific Software as Workflows: From Discovery to Distribution," IEEE Software, vol. 25, no. 4, pp. 37-43, July/Aug. 2008
[5] Ewa Deelman, Dennis Gannon, Matthew Shields and Ian Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities", Future Generation Computer Systems Volume 25, Issue 5, May 2009
[6] Yogesh Simmhan, Roger Barga, Catharine van Ingen, Ed Lazowska, Alex Szalay, "Building the Trident Scientific Workflow Workbench for Data Management in the Cloud", ADVCOMP 2009 (to appear).
[7] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, "A Note on Distributed Computing", SMLI TR-94-29, Sun Micro., 1994.
[8] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System", Symposium on Operating Systems Principles, 2003.
[9] Pat Helland, Dave Campbell, "Building on Quicksand", CIDR 2009.
[10] N. Kaiser, "The Pan-STARRS Survey Telescope Project", Bull. American Astronomical Society, 37 (4), 2006.
[11] Roger Barga, Jared Jackson, Nelson Araujo, Dean Guo, Nitin Gautam, and Yogesh Simmhan, "The Trident Scientific Workflow Workbench", in IEEE eScience Conference, 2008.
[12] Alexander Szalay, et al., "GrayWulf: Scalable Clustered Architecture for Data Intensive Computing" HICSS, 2009.
[13] Yogesh Simmhan, et al., "GrayWulf: Scalable Software Architecture for Data Intensive Computing", HICSS, 2009
[14] Ž. Ivezić, et al., "Astrometry with digital sky surveys: from SDSS to LSST", International Astronomical Union (2007), 3:537-543.
[15] Albrecht, M. A. & Heck, A., "StarGates. A database of astronomy, space sciences, and related organizations of the world", Astron. Astrophys. Suppl. 103, 473-474 (1994).
[16] N. Li and A.R. Thakar, "CasJobs and MyDB: A Batch Query Workbench," Computing in Science & Eng., vol. 10, no, 1, 2008, pp. 18–29.
[17] A. Szalay, "The Sloan Digital Sky Survey," Computing in Science and Engineering, vol. 1, no. 2, pp. 54-62, Mar./Apr. 1999
[18] Ping Yang, "Formal Modeling and Analysis of Scientific Workflows Using Hierarchical State Machines", SWBES, 2007
[19] Meichun Hsu Ron Obermarck, Roelof Vuurboom, "Workflow Model and Execution", Data Engg. Bulletin, 16(2) 1993.
[20] R. Duan, R. Prodan, T. Fahringer, "DEE: A Distributed Fault Tolerant Workflow Enactment Engine", LNCS 3726/2005.
[21] IBM Business Flow Manager, http://www.ibm.com/software/info/bpm/
[22] David Chappell, "Introducing Microsoft Windows Workflow Foundation: An Early Look", MSDN Article, August 2005.
[23] Slomiski, A. "On using BPEL extensibility to implement OGSI and WSRF Grid workflows", Concurr. Comput. : Pract. Exper. 18 (10), 2006.
[24] J Gray, DP Siewiorek, "High-availability computer systems", IEEE Computer, 1991
[25] Yogesh Simmhan, "End-to-End Scientific Data Management Using Workflows", SWF 2008
[26] Anand, M. K., Bowers, S., McPhillips, T., and Ludäscher, B. "Efficient provenance storage over nested data collections". EDBT, Vo. 360, 2009.
[27] Yogesh Simmhan, Beth Plale and Dennis Gannon, "A Survey of Data provenance in eScience", Sigmod Record 34(3), 2005.
[28] Satya S. Sahoo, Amit Sheth, and Cory Henson, "Semantic Provenance for eScience: Managing the Deluge of Scientific Data", IEEE Internet Computing 2008.
[29] J Zhao, C Wroe, C Goble, R Stevens, Dennis Quan and Mark Greenwood, "Using Semantic Web Technologies for Representing E-science Provenance", ISWC 2004
[30] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana Maclean, Daniel Margo, Margo Seltzer, and Robin Smogor, "Layering in Provenance Systems", USENIX, 2009
[31] James Frew, Peter Slaughter: "ES3: A Demonstration of Transparent Provenance for Scientific Computation". IPAW 2008
[32] Walid and Claude Godart, "Mining Workflow Recovery from Event Based Logs", LNCS 3649, 2005.
[33] Yi Huang, Aleksander Slominski, Chathura Herath, Dennis Gannon, "WS-Messenger: A Web Services-Based Messaging System for Service-Oriented Grid Computing," CCGrid, 2006.
[34] RD Stevens, AJ Robinson, CA Goble "myGrid: personalised bioinformatics on the information grid", Bioinformatics, 2003
[35] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, Steven Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets", Journal of Network and Computer Applications, 1999.
[36] Randall Bramley, et al. "Instrument Monitoring, Data Sharing, and Archiving Using Common Instrument Middleware Architecture (CIMA)", J. Chem. Inf. Model., 2006, 46 (3)
[37] D.A. Patterson, G. Gibson, and R.H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," SIGMOD, 1988.
[38] Kola, G., Kosar, T., and Livny, M. "Phoenix: Making Data-Intensive Grid Applications Fault-Tolerant", Grid 2004.

[39] Ann L. Chervenak, Naveen Palavalli, Shishir Bharathi, Carl Kesselman, Robert Schwartzkopf, "Performance and Scalability of a Replica Location Service," HPDC, 2004.
[40] Philip A. Bernstein, Nathan Goodman, “The failure and recovery problem for replicated databases”, Symposium on Principles of Distributed Computing, 1983.
[41] Kandaswamy, G.; Mandal, A.; Reed, D.A. “Fault Tolerance and Recovery of Scientific Workflows on Computational Grids”, CCGrid, 2008.
[42] Ian Foster, Jens Vöckler, Michael Wilde, Yong Zhao, “Chimera: A Virtual Data System For Representing, Querying, and Automating Data Derivation”, SSDBM 2002.
[43] Crawl, D. and Altintas, I. “A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows”, IPAW 2008.
[44] Mehul A. Shah, Joseph M. Hellerstein, Eric Brewer, “Highly Available, Fault-Tolerant, Parallel Dataflows”, SIGMOD 2004.
[45] Grefen, P. W. “Transactional Workflows or Workflow Transactions?”, Database and Expert Systems Applications, 2002.
[46] R Barga, D Gannon, “Scientific versus Business Workflows”, Workflows for E-Science: Scientific Workflows for Grids, 2006
[47] Roger S. Barga, David B. Lomet: Phoenix Project: Fault-Tolerant Applications. SIGMOD Record 31(2): 94-100 (2002)
[48] Skeen, D. and Wright, D. D. “Increasing availability in partitioned database systems”. Principles of Database Systems, 1984.
[49] Luc Moreau, et al. “The Open Provenance Model (v1.01)”, Tech. Report 16148, Univ. of Southampton 2008.
[50 Valerio, M.D.; Sahoo, S.S.; Barga, R.S.; Jackson, J.J., "Capturing Workflow Event Data for Monitoring, Performance Analysis, and Management of Scientific Workflows," eScience, 2008.
[51] My Experiment web portal, http://www.MyExperiment.org
[52] David DeWitt, Jim Gray, “Parallel database systems: the future of high performance database systems”, Communications of the ACM, 35(6), 1992
[53 Armando Fox, Eric A. Brewer, "Harvest, Yield, and Scalable Tolerant Systems," Hot Topics in Operating Systems, pp.174, 1999
[54] SQL Server Replication, http://msdn.microsoft.com/en-us/library/ms151198.aspx
[55] Microsoft Sync Framework, http://msdn.microsoft.com/en-us/sync/bb821992.aspx
[56] RAID 1+0, http://en.wikipedia.org/wiki/RAID
[57] María A. Nieto-Santisteban, Aniruddha R. Thakar, and Alexander S. Szalay, “Cross-Matching Very Large Datasets”, NASA Science Technology Conference (NSTC), 2007.