THESIS

A LOCALITY-AWARE SCIENTIFIC WORKFLOW ENGINE FOR FAST-EVOLVING SPATIOTEMPORAL SENSOR DATA

Submitted by

Johnson Charles Kachikaran Arulswamy

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2017

ABSTRACT

A LOCALITY-AWARE SCIENTIFIC WORKFLOW ENGINE FOR FAST-EVOLVING SPATIOTEMPORAL SENSOR DATA

Discerning knowledge from voluminous data involves a series of data manipulation steps. Scientists typically compose and execute workflows for these steps using scientific workflow management systems (SWfMSs). SWfMSs have been developed for several research communities including but not limited to bioinformatics, biology, astronomy, computational science, and physics. Parallel execution of workflows has been widely employed in SWfMSs by exploiting the storage and computing resources of grid and cloud services. However, none of these systems have been tailored for the needs of spatiotemporal analytics on real-time sensor data with high arrival rates. This thesis demonstrates the development and evaluation of a target-oriented workflow model that enables a user to specify dependencies among the workflow components, including data availability. The underlying spatiotemporal data dispersion and indexing scheme provides fast data search and retrieval to plan and execute computations comprising the workflow. This work includes a scheduling algorithm that targets minimizing data movement across machines while ensuring fair and efficient resource allocation among multiple users. The study includes empirical evaluations performed on the Google cloud.

ACKNOWLEDGMENTS

First and foremost, I thank God for giving me strength and emotional stability during this project, without which I could not have finished this work successfully.

I would like to express my heartfelt gratitude to so many people for their help and support in one way or the other, making this study possible.

To my parents, Valli and Roopan, and my sister, Rajkumari, I thank you deeply for your constant support and encouragement.

I would like to express my gratitude to my longtime friends, especially to Sreenivas Potukuchi, Shashi Kiran Peddala, Mohammed Amer, Sandeep Donthula, Harsha Vannemreddy, and Sandeep Komaravelly, in no particular order, for always being there for me sharing my joy and sorrow. Special thanks to Sandeep Donthula, who was the reason for my travel here, Mohammed Amer for his support to my family in my absence, and Harsha Vannemreddy for his active help in this work.

I would also like to extend my gratitude and acknowledgment to my committee members, Dr. Sangmi Lee Pallickara, Dr. Shrideep Pallickara, and Dr. Joseph von Fischer for their valuable insights and feedback. I would like to express special thanks to Dr. Sangmi Lee Pallickara and Dr. Joseph von Fischer for giving me an opportunity to work with them, particularly to Dr. Sangmi Lee Pallickara for her constant guidance and support in making this work successful. Greatest thanks also to the Environmental Defense Fund, who funded this study, and Karin Tuxen-Bettman of Google Inc. for productive discussion and feedback through this collaborative research, and for providing computing environment to us.

Many thanks to Naman Rajivkumar Shah and Harshil Ketanbhai Shah for being good friends. I really appreciate your support during this work. Special thanks to Naman Shah for the countless number of hours spent working together day and night in CSB435, innumerable cups of coffee, for all the patience it took to go through my practice talk repeatedly, and for all the arrangements he made for my defense. On a different note, Naman, we should really reduce our coffee consumption.

I am tremendously grateful to Dr. Lumina Albert, for all the mentoring, guidance, and support. Dr.Albert, thank you so much for attending my defense.

Last but not least, I would like to thank Dr. Deborah Parker, for taking time to edit and proofread this manuscript.

This thesis is typset in LaTeX using a document class designed by Leif Anderson.

DEDICATION

To my parents

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Proliferation of large-scale wireless geo-sensor networks enables continuous and precise capture of large amounts of geospatial, time-series information. Examples of such spatiotemporal data harvesting can be found in settings ranging from nuclear power plants (nuclear optical sensors) to smartphones (equipped with myriad sensing capabilities). GPS-enabled sensors, either static or on mobile platforms, collect information frequently, and these data are used as inputs to analytical processes that subsequently inform the monitoring environment. As these spatiotemporal data collections become more available, the ability to generate insights from sensor datasets has been critical for many high-priority scientific applications.

One of the dominant approaches to implementing complex scientific analyses is to modularize the computing aspects. Each module then serves as a stage within a pipeline. One or more such pipelines comprise a Scientific Workflow Management System (SWfMS). A SWfMS can involve heterogeneous programming languages and tools. Furthermore, tasks within a SWfMS can be implemented in domain-specific tools that were developed for use within the scientific community. Consequently, scientists in several domains have widely adopted SWfMSs to facilitate composition, execution, monitoring, and sharing of such complex computing components.

Much of the research in SWfMSs has emphasized usability aspects including ease of use, composition, and reusability of software components[1–3]. Pegasus[4] and Askalon[5] target effective execution in a distributed environment. Recent systems, such as Apache Spark and Apache Flink/Stratosphere, adapt the MapReduce programming model to express stages and connections thereto. These efforts are not well-aligned with processing real-time sensor data collected across the globe at different points in time that are not necessarily uniform or periodic.

Analytic workflows for the aforesaid datasets are quite distinctive compared to traditional scientific workflows. This stems from the fact that such sensor data processing involves: (1) data selection constrained by arbitrary space and time, and (2) volatile resource requirements. We define these workflows as *data-bounded workflows* whose stages account for the availability of data, both input and intermediate. A workflow engine capable of executing a data-bounded workflow must have the data selection mechanism integrated into the system, and it should be able to meet the requirements of such workflows. The objective of this thesis is to support the composition and orchestration of data-bounded workflows in the context of fast-evolving voluminous real-time datasets generated in sensing environments.

1.1. Scientific Challenges

Efficient composition, scheduling, and execution of scientific workflows over voluminous sensor data streams are tasks that introduce a set of unique challenges:

- **Voluminous data**: As the number of sensors grows, cumulative data generated by the sensor network increases dramatically.
- **Real-time data analysis**: Despite the number of interacting components, their aggregations, and the rates at which data arrive, users must be able to analyze the results, both intermediate and final, in real time.
- **Shared computing and storage resources**: Orchestration and execution of the workflow must be possible in a shared cluster with collocated processes and their accompanying resource footprints.
- **Scalability**: The proposed approach must scale with increases in the number of workflows and data volumes.

1.2. Research Questions

Inability to account for data characteristics within the workflow may result in queue buildups, buffer overflows, increased latencies, and possibly reduced throughputs. Research questions that are explored in this thesis include:

- RQ-1 How can we compose data-bounded workflows? To allow users to compose data-bounded workflows, the workflow must be able to identify data-bounded workflow components and their corresponding dependencies alongside their mutable states.
- RQ-2 How can we disperse spatiotemporal sensor data to support efficient data retrievals over fast-evolving time-series phenomena? The data dispersion and indexing scheme should provide fast data search and retrieval, and it must also allow the workflow engine to allocate tasks with high data locality.
- RQ-3 How can we schedule and execute data-bounded workflows effectively in high-throughput analysis environments? The workflow engine should be able to schedule and execute realizable portions of the analysis to allow users to monitor incremental, intermediate results for long-running analytics tasks in real time. It must accomplish this without postponing workflow execution until all data is available or preempting computing resources.

1.3. Overview of our Approach

We introduce a locality-aware distributed workflow engine called Columbus that schedules and executes data-bounded workflows while accounting for data locality and fair resource allocation among multiple users. Columbus uses Galileo[6], a hierarchical distributed hash table, as its underlying storage system. We extended Galileo to enable a flexible data dispersion and indexing scheme for the space-time continuum such that data locality is maximized during the orchestration of computations. To facilitate the composition of complex data-bounded workflows, we propose a target-oriented workflow model that not only addresses the traditional SWfMS aspects such as composing workflows with ease and reusability of the workflow components but also accounts for data availability to individual components of the workflow. Columbus manages the execution of these workflows using a *master-worker* architecture that employs a dual scheduling strategy where both master and the workers run a scheduler to exploit parallelism at workflow and task levels. We have devised three locality-aware scheduling schemes, *local*, *remote*, and *hybrid*, in order to slate the workflows for execution. The *local* scheme ensures the highest data locality, while the *remote* scheme allows the tasks to be reallocated based on the resource utilization. The *hybrid* scheme is a balance between *local* and *remote* that relies on *WR ratios* obtained from the computing nodes, a metric representing the ratio of number of workflows waiting to running per user.

1.4. Thesis Contributions

This study describes a method that achieves efficient management of analytical scientific workflows over rapidly evolving sensor data. Specific contributions include:

(1) An effective data dispersion and indexing scheme that maximizes data locality to avoid data movements and resource contentions.

(2) An expressive workflow composition model that enables complex spatiotemporal analyses with effective planning and tracking.

(3) Our methodology encompassing algorithms and data structures could also be applied to other scientific workflow engines that involve both resource- and data-constrained scheduling.

Our empirical evaluations demonstrate the feasibility of our approach and its ability to scale. In our benchmarks, the test dataset spans more than 25 million observations, each of which contains 40 features for a specific location. Unless otherwise noted, the location information of the data we considered was altered for the purpose of our experiments and illustrations included in this work. Our benchmarks also contrast the data retrieval performance of Columbus with Geomesa[7].

CHAPTER 2

Related Work

Several scientific workflow management systems exist[8, 9] to help discover knowledge from the vast information at hand that have been developed for various research communities, including but not limited to bioinformatics, astronomy, biology, computational engineering, and earth sciences. Yet, new systems emerge from time to time because it is unlikely that any single system can handle such diverse domain needs[10].

Kepler[1] is a free and open-source scientific workflow application designed to help scientists, analysts, and computer programmers create, execute, and share models and analyses across a broad range of scientific and engineering disciplines. Kepler can operate on data stored in a variety of formats, locally and over the Internet, and it is an effective environment for integrating disparate software components, for example, merging R scripts with compiled C code or facilitating remote, distributed execution of models. Using Keplers graphical user interface, users simply select and then connect pertinent analytical components and data sources to create a scientific workflow.

Taverna[2] is also an open-source and domain-independent SWfMS used to design and execute scientific workflows and aid *in silico* experimentation. It was created by the *myGrid* team, and it is now an Apache Incubator project. The Taverna tools include the Workbench (desktop client application), the Command Line Tool (for a quick execution of workflows from a terminal), the Server (for remote execution of workflows), and the Player (Web interface plugin for submitting workflows for remote execution). Both Kepler and Taverna allow users to share workflows through myExperiment[11], a collaborative environment where scientists can safely publish their workflows and *in silico* experiments.

Triana[12] is an open-source problem-solving environment developed at Cardiff University that combines an intuitive visual interface with powerful data analysis tools. Already used by scientists for a range of tasks, such as signal, text, and image processing, Triana includes a large library of pre-written analysis tools and the capability for users to easily integrate their own tools. Triana is particularly good at automating repetitive tasks, such as performing a find-and-replace on all of the text files in a particular directory or continuously monitoring the spectrum of data that comes from an experiment that runs for days or even years.

Galaxy[3] is another scientific workflow system that allows users to import workflows from myExperiment, but this system was specifically developed for data-intensive biomedical research. However, Galaxy uses Gridway to execute tasks on the grid, and it can exploit Globus[13] and CloudMan[14] to achieve dynamic computing and storage provisioning across computing nodes.

The Pegasus project[4] encompasses a set of technologies that help workflow-based applications execute in a number of different environments including desktops, campus clusters, grids, and clouds. Pegasus bridges the scientific domain and the execution environment by automatically mapping high-level workflow descriptions onto distributed resources. It automatically locates the necessary input data and computational resources necessary for workflow execution. Pegasus enables scientists to construct workflows in abstract terms without worrying about the details of the underlying execution environment or the particulars of the low-level specifications required by the middleware (Condor, Globus[13, 15], or Amazon EC2). Pegasus also bridges the current cyberinfrastructure by effectively coordinating multiple distributed resources.

The ASKALON project[5] crafts a novel environment based on new innovative tools, services, and methodologies to make scientific application development and optimization for real applications and execution on Cloud and Grid environments an everyday practice. Askalon is designed as a distributed service-oriented architecture that provides a rich set of visualization diagrams for post-mortem and online visualization.

Script-based workflows such as Swift[16] and JS4Cloud[17] exist as alternatives to the visual workflows discussed above. They can provide implicit data-driven task parallelism and data parallelism. The former defines a new parallel scripting language to model the workflows that follows C-like syntax, and the latter extends JavaScript for the same.

None of these systems are tailored for the needs of data-bounded scientific workflows with continuously arriving sensor data. Columbus addresses such workflows with its collocated scalable storage system and target-oriented workflow modeling, which allow users to specify complex workflows having volatile resource requirements with an integrated data selection mechanism.

Researchers have also explored many-task computing (MTC)[18] in scientific workflows[19]. Ogasawara et al. aim at providing a middleware solution as a bridge between SWfMSs and high performance computing (HPC), supporting workflow design and provenance combined to MTC. Frameworks such as Falkon[20], SWARM[21], and AME[22] aim at rapid execution of many tasks on distributed computing clusters and grids. The intention was to achieve a shorter *makespan*. Wang et al. introduced load-balanced and locality-aware scheduling for data intensive workloads at extreme scales through data-aware work stealing(DAWS) technique[23]. A survey of the workflow scheduling algorithms[24] reveals that much of the research was in makespan optimization. However, in this work, we aim at the execution of workflows with high data locality and fair resource allocation among multiple users. The scheduling mechanism must also ensure reliable workflow execution given the fact that the memory requirements of the tasks involved in a data-bounded workflow are volatile.

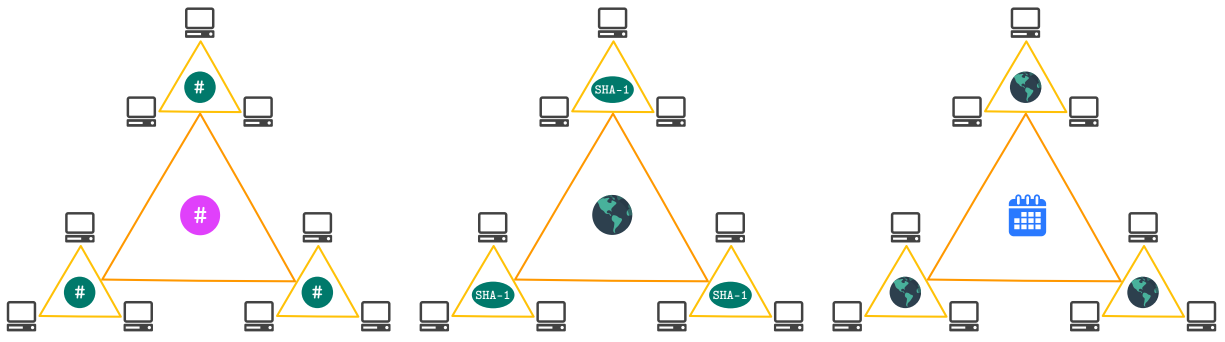
CHAPTER 3

Storage Framework

For our storage requirements, we extended Galileo[6, 25–27], a high-throughput, distributed storage framework for voluminous, multidimensional, geospatial, and time-series datasets. We modified the data dispersion scheme to maximize the data locality and to cope with continuously arriving sensor data streams. In this chapter, we discuss the extensions made to Galileo in detail.

3.1. Data Partitioning

Galileo is a zero-hop hierarchical distributed hash table (DHT) where nodes are organized into groups such that each node has enough information about the network topology to route requests directly to their destinations[6]. In the original partitioning scheme of Galileo, a group of nodes was determined by Geohash[28], whose length indicated its precision, and a node within the group was identified by SHA-1 hash. In this work, we reoriented the original partitioning scheme to cope with unevenly distributed time-series sensor datasets by considering a temporal hash function. A destination group is determined first by using a hash scheme based on the temporal information of the data, and then Geohash is used to determine the storage node within a group. Fig. 3.1 shows the network organization of the hierarchical DHT (3.1a), the original geospatial partitioning scheme (3.1b), and the proposed chrono-spatial partitioning scheme (3.1c).



(a) Hierarchical DHT *(b) Geospatial DHT* *(c) Chronospatial DHT*

Figure 3.1. Network Organization

The new Galileo partitioning mechanism provides a configurable temporal hash scheme, allowing the user to choose a temporal pattern such as hour, day, week, month, or year for hashing the data to a group based on time. The choice is generally made based on how a user wishes to process the data, which could be different from how the data is getting collected. For instance, wind data is collected by the National Climatic Data Center (NCDC) of the National Oceanic and Atmospheric Administration every hour, but the climatological mean wind speed is reported every month[29]. In such cases, choosing *month* as the temporal hash type will ensure that the data of any month for a particular geospatial region is available with the minimum number of nodes. This reduces data movement during computation significantly.

Unlike the original system, which was developed to store a single dataset and where the partitioning scheme must be chosen beforehand, the new extension uses a dynamic network organization so that the partitioning scheme can be specified at the time of creation of the dataset. This feature allows a user to store multiple datasets with different partitioning schemes, which is another important refinement made to the system because scientific workflow analyses often deal with more than one dataset. Fig. 3.2 shows the base network organization having three groups followed by a dataset having a chrono-spatial partitioning scheme with three groups temporally partitioned on *day of month*, a second dataset with the same partitioning scheme as before but with five groups, and a third dataset having a chrono-spatial partitioning scheme with nine groups temporally partitioned on *week of year*.

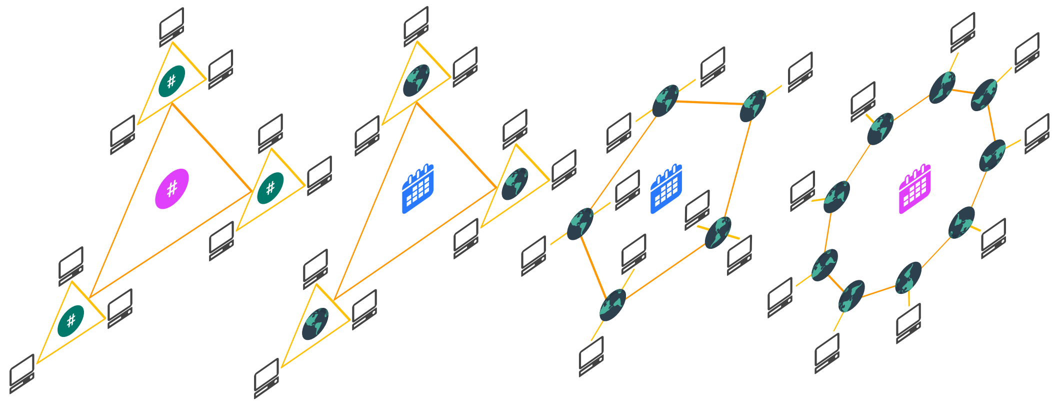


Figure 3.2. Dynamic Network Organization

The proposed chrono-spatial partitioning scheme can also be turned into a spatiotemporal partitioning scheme where a group would be determined by Geohash and a node by the temporal information of the data. The choice of this scheme depends on the dataset and the need for parallelism at spatial or temporal levels. We adhere to the chrono-spatial scheme in this work as it is more appropriate for the dataset we considered.

3.2. Data Storage and Retrieval

Galileo was intended to store point datasets, meaning the data associated with a spatial location is stored in the system. While data of a spatial region can be stored in the system, it is represented internally by the spatial location of the centroid of the spatial region. The storage units are called as *blocks*, and they are stored as files on the host file system. Any information representing the block is stored in a hierarchical graph residing in the memory, referred to as the *metadata graph*. Also, the retrieval process was devised to differ from traditional databases or key-value stores. Instead of matching the user submitted queries against the data available in the system and returning the raw data, it streams metadata of matching blocks back to the requestor incrementally. This approach works fine when the user is interested in retrieving the blocks; however, this may not always be the case for scientific analyses. If a user makes a spatial query and is interested in the raw data, then the blocks returned by Galileo should undergo further processing at the client side.

With the refinements made to the Galileo framework, data of a spatial region can be stored in the system with user-defined Geohash precision that can be different from the Geohash precision used for the spatial partitioning scheme. This gives the user more flexibility in grouping the data together for storage in Galileo. Furthermore, a user can specify the features of the dataset at the time of its creation in the system and designate the features responsible for the spatial location as a hint. This enables Galileo to construct another hierarchical graph similar to the metadata graph from the information stored in the block, called as *feature graph*, and return the raw data to the user when requested. As a consequence of this change, the system distinguishes the queries made as *metadata queries* and *feature queries* depending on whether the request was made for blocks or raw data, respectively. However, a user can specify both metadata and features in the same query request when seeking to retrieve the raw data.

For the purpose of understanding, consider the NYC Yellow Taxi Dataset[1] that has 19 features.[2] As the dataset is confined to the New York city region, it is a good idea to consider a spatial partitioning scheme having Geohash of length three involving a subset of the regions in the Geohash area of `dr`, specifically `dr4`, `dr5`, `dr6`, `dr7`, `drk`, and `drh`. If we decide to store the trips that started within an area of 3miles x 3miles per day together, which can be represented by a Geohash of length five, and use the attributes `min_amount`, `mean_amount`, `max_amount`, and `num_trips` to represent the minimum, mean, maximum total amounts, and number of trips in that data, respectively, then an extract of the metadata graph on the node having the Geohash `dr5` is shown in fig. 3.3, and an extract of the feature graph for the block `2015-12-31-dr5ru.gblock` is shown in fig. 3.4.

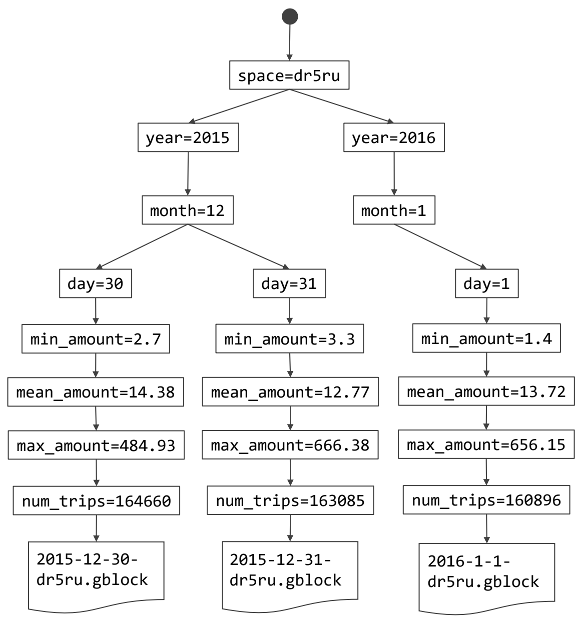


Figure 3.3. Metadata Graph

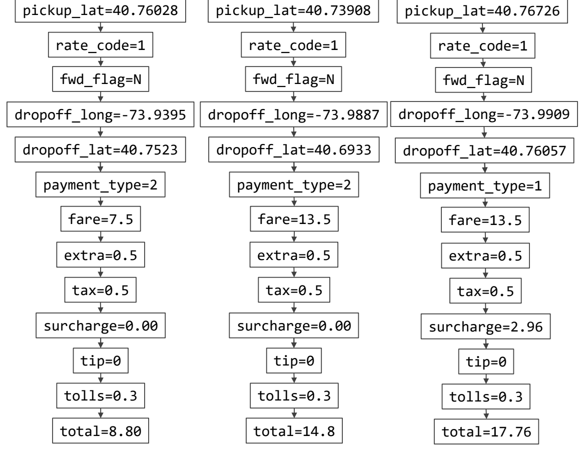


Figure 3.4. Feature Graph

[1]http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
[2]http://www.nyc.gov/html/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf

When a user wants to retrieve the blocks having `mean_amount` more than \$500, a metadata query (`mean_amount > 500`) is issued to all of the nodes in the system, and the resulting blocks are returned to the client. However, if the user is interested in the trips that cost more than \$50, then a feature query (`total > 50`) is issued. Alternatively, if the user is interested in the trips that cost more than \$50 on a particular day, say `2015-12-31`, both metadata and feature queries are made in the same request (e.g., `year=2015 & month=12 & day=31, total > 50`).

3.3. The Geoavailability Grid

A *geoavailability grid* is a spatial indexing data structure[25] that translates points in space to a reduced-resolution coordinate system for indexing purposes. It is described by a bitmap denoted by a vector of bits, where a bit is set to 1 if information has been stored in that location and set to 0 otherwise. Galileo uses this grid to evaluate the spatial queries such that the polygonal bounds of the query are decomposed into smaller polygons that cover the area of the grid. A new bitmap is generated for the resulting query and is intersected with the bitmap representing the data. The points that are set to 1 in both bitmaps are returned. Fig. 3.5 shows an example data bitmap in blue, a query bitmap in red, and the result of the intersection in yellow.

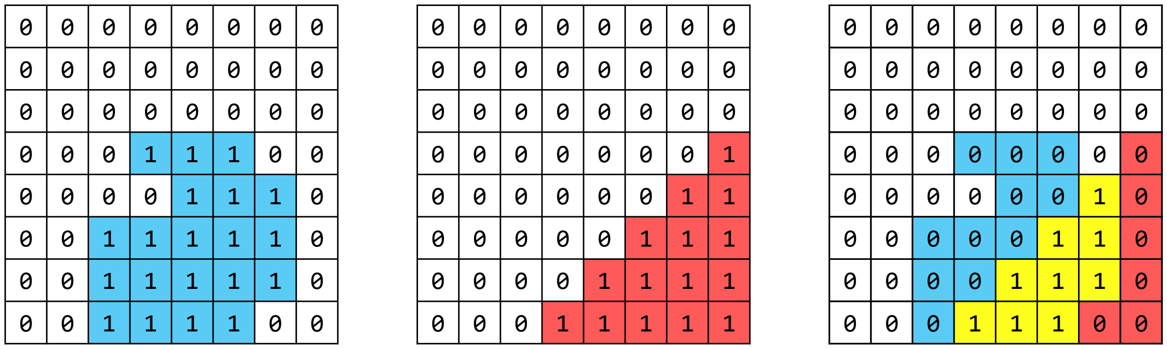


Figure 3.5. Geoavailability Grid Evaluation

Unlike the original system, where a grid is maintained at a global level, we create a 30-bit resolution grid on demand for the necessary blocks representing the area covered by them. If the blocks are stored with a precision of four characters, or 20 bits, the actual resolution of the grid can be seen as 50-bit, which gives accuracy to nearly one meter in space. Moreover, the on-demand creation of the grid aids in making the Galileo system collocated with the Columbus workflow engine.

3.4. Query Evaluation

Galileo supports a variety of queries including feature queries, range queries, and geospatial retrievals constrained by arbitrary polygonal bounds. The new partitioning scheme allows the system to efficiently address the spatial and temporal queries with or without filtering criteria specified on the features of the dataset or the metadata of the blocks, simply *filters*.

3.4.1. Spatial Queries. When a spatial query such as a polygon is submitted to Galileo, the system tries to identify the nodes that can address the query. It starts the process by looking up the precision level set for the spatial partitioning scheme. A Geohash with that precision is computed for any one vertex of the polygon, and its bounding box is inspected to determine if the polygon was enclosed by the box. In such a case, the process terminates, and the query is directed to the nodes responsible for that Geohash. Otherwise, the system computes the set of all neighboring Geohash values and determines if their bounding boxes are intersecting with polygon. The process is repeated, eliminating the non-intersecting Geohash values until no more intersections are found. The set of destination nodes from the resulting set of intersecting Geohashes is obtained from the spatial partitioning scheme, and the query is directed to those nodes. Fig. 3.6 shows a spatial query made in the region of the State of Colorado. A textual representation of this query translates to:

```
GET ALL FEATURES FROM DATASET IN POLYGON("-105.27 39.74,-105.01 39.87,-104.63
39.76,-104.57 39.52,-104.91 39.32,-105.23 39.43")
```

With a two-character Geohash precision set for the spatial partitioner and a four-character Geohash precision set for the blocks, the system starts with one of the vertices of the polygon (e.g., one in 9xj6) and constructs the Geohash value as 9x. It computes all eight neighbors of this Geohash as shown in fig. 3.7. The intersecting Geohashes are highlighted in bold. Of the neighbors, only 9w is found to be intersecting, and the system recomputes its neighboring Geohashes, excluding the ones already processed. The resulting set is used to identify the set of destination nodes.

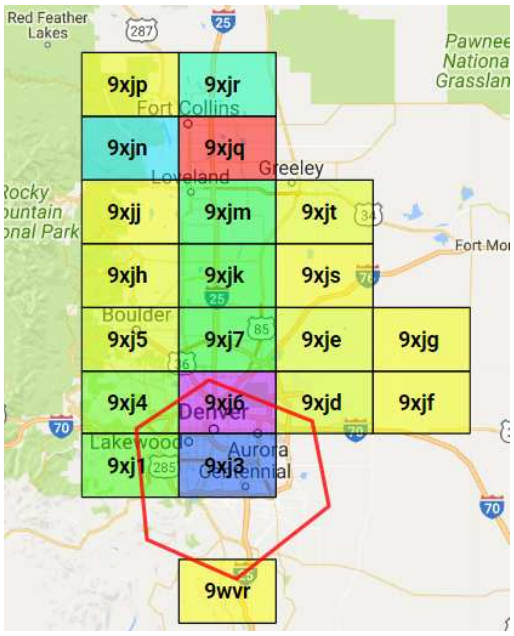


Figure 3.6. Columbus showing the data reported by Galileo in the State of Colorado and a spatial query made around Denver, CO

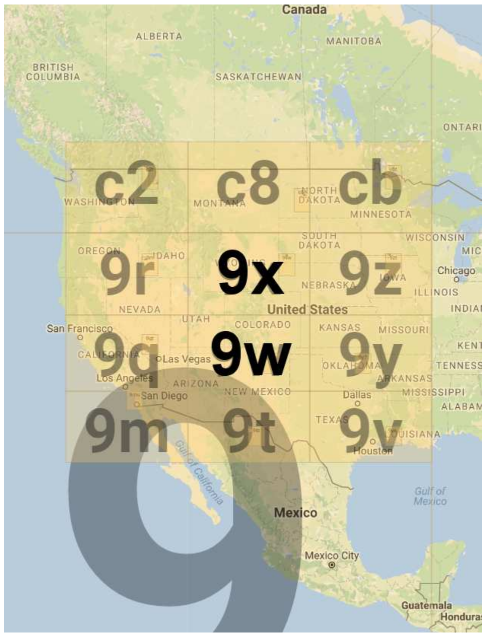


Figure 3.7. Determining intersecting Geohash values to direct the spatial query to the destination nodes

At the destination nodes, the system looks up the Geohash precision of the dataset and computes the intersecting Geohash values of that precision. Unlike the process of identifying nodes, the system uses a *recursive binary partitioning* on each Geohash region present at the destination that is known to intersect the spatial query. This process returns a set of range queries on the spatial attribute of the metadata graph. If a filter was specified on the metadata, the metadata query together with the computed range queries are evaluated on the metadata graph to obtain the set of blocks that contain the data for the region specified by the spatial query. For each block, the system loads the raw data into a Geoavailability grid if the spatial region of the block intersects the query based on the spatial hint provided at the time of creation of the dataset. Otherwise, the spatial region of the block is enclosed by the query and computation on the grid can be avoided. The grid evaluates the raw data against the query and returns the results. If a filter was specified on the features of the dataset, the returned results are then turned into a *feature graph*, and the query is evaluated on it. The final results are written to the disk so that the Columbus workflow engine can push the computation to that storage node.

For the query made in fig. 3.6, the recursive binary partitioning approach on the `9x` Geohash region returns a set of range values that result in the following query:

```
GET ALL FEATURES FROM DATASET WHERE (space≥9xj0 and space≤9xj3) or space=9xj4
        or space=9xj6  or (space≥9xj8 and space≤9xj9) or space=9xjd
```

This query is then evaluated on the metadata graph to retrieve the blocks in those regions, in this case `9xj1`, `9xj3`, `9xj4`, `9xj6`, and `9xjd`. The system identifies that the region `9xj3` was enclosed by the query and considers the blocks in that region as evaluated. For the blocks in the other regions, data is loaded into the Geoavailability grids and evaluated against the spatial query intersecting that region. Fig. 3.8 shows the grids for those regions, highlighting the spatial query in red and the data in blue.

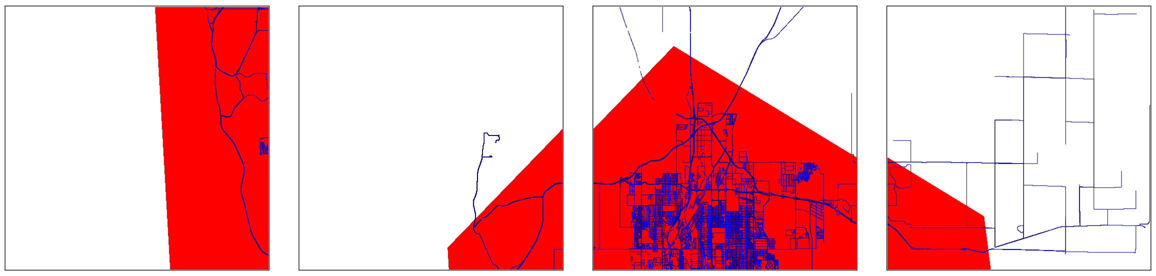

(a) 9xj1 *(b) 9xj4* *(c) 9xj6* *(d) 9xjd*

Figure 3.8. Geoavailability grids for the query in fig. 3.6 at nodes having 9x Geohash

3.4.2. *Temporal Queries.* Queries made in accordance with the temporal type specified for the temporal partitioning scheme are directed to the nodes involved in the matching group. Otherwise, they are treated as a metadata filter and are directed to all nodes in the system. For instance, if a dataset was partitioned temporally on *day of the month* and the query includes the day, the temporal partitioning scheme identifies a group responsible for having the data for that day, so the query is directed to all of the nodes of that group. However, if the query was made on the *month of the year* for the same dataset, the temporal partitioning scheme cannot identify the group; hence, the query is directed to all nodes of all groups. In any case, queries are evaluated on the metadata graph at their respective destinations. If a filter was specified on the metadata, it would be included in the query being evaluated on the metadata graph. The resulting blocks obtained from the metadata graph do not need any processing if a filter was not specified on the features of the dataset, and they can be returned to the requestor. Otherwise, data from the blocks are loaded into feature graphs and evaluated against the feature query. The final results are written to the disk.

3.4.3. *Spatiotemporal Queries.* Queries made on both space and time are resolved first by time followed by space to identify the destinations. The first phase identifies the groups based on the temporal type specified for the temporal partitioning scheme. The second phase eliminates the nodes based on the spatial query as discussed in section 3.4.1. At the destinations, the recursive binary partitioning technique yields the range queries on space, and they are evaluated on the metadata graphtogether with the metadata query on time and the metadata filters if anyto obtain the blocks. The resulting blocks are then evaluated on Geoavailability grids, and the results are written to the disk if a filter was not specified on the features of the dataset. Otherwise, they are loaded into feature graphs and evaluated against the feature query, and the final results are written to the disk.

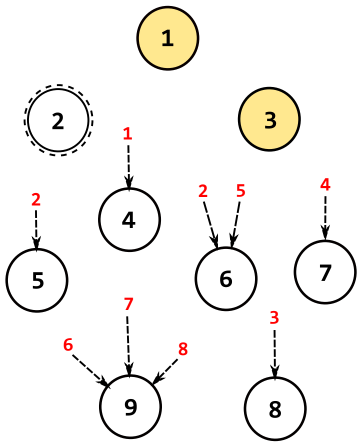
CHAPTER 4

System Design

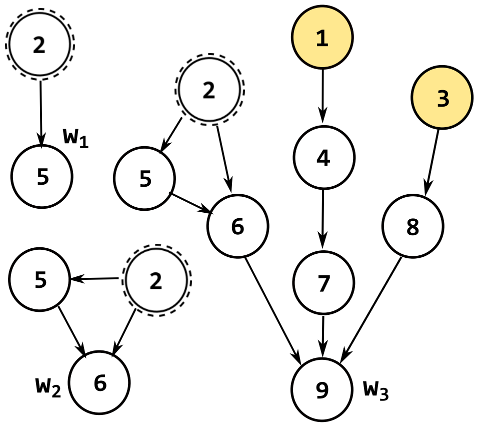
A workflow engine capable of providing insights on vast spatiotemporal datasets must have its elements aligned to such domain needs. In this chapter, we discuss the core design details of the system that aid in doing such analyses.

4.1. Target-Oriented Workflow Model

Designing a complex data-bounded workflow is a challenging task, and Columbus addresses this concern with a *target-oriented modeling* paradigm. The system can be viewed as a composition of interdependent atomic units, called *targets*. A scientist defines a target by specifying its dependencies on other targets in the system, hereafter referred to as *parents*, and optionally wraps a computation in it, typically as a sequence of high-level programming instructions. These dependencies are defined as *active* if the execution of a target invokes the execution of its parents, and they are *passive* if the execution of a target depends on the output of the parents but does not invoke their execution. Every target is associated with one of the output types supported by the system, and this includes a visualizer to make the system enable data visualization by means of *web mapping*[30]. Target-oriented modeling then allows the composition of a workflow by means of a weak association between the targets and the workflow, meaning the targets are not coupled with the workflow that encapsulates them. Rather, any target from the pool can be turned into a workflow by tracing back its dependencies. This design principle is the key to leveraging reusability of targets across workflows, and it represents a workflow as a directed acyclic graph of targets. A distinctive feature arising out of this design principle is that it allows a scientist to choose data from multiple data sources for a single workflow. Fig. 4.1a captures a few targets defined in the system, and some workflows composed from those targets are shown in fig. 4.1b. We denote targets with circles, active dependencies with a dashed line, passive dependencies with a dashed circle, and data flow with a solid line. A workflow is denoted as a directed acyclic graph of targets connected by a solid line.



(a) Targets



(b) Workflows

Figure 4.1. Modeling Workflows from Targets

4.2. Building Blocks

Targets are realized in the system as Components and Combiners. A Component defines a set of active dependencies on other Components and Combiners in the system. This set can be null, meaning that the Component is independent. We refer to Components without dependencies as $\phi-$Components, those with one active dependency as $\alpha-$Components, and those with multiple active dependencies as $\beta-$Components. Graphically, $\alpha-$ and $\beta-$Components are denoted by circles with a solid line, and $\phi-$Components are denoted by filled circles. Fig. 4.1a shows eight Components, of which 1 and 3 are $\phi-$Components, 4, 5, 7, and 8 are $\alpha-$Components, and 6 and 9 are $\beta-$Components. Each Component may define a sequence of high-level programming instructions to process the data that flows through it and must define an output.

Any Component can be turned into a Workflow, which can be seen merely as an entity that facilitates the flow of data among the targets. A Workflow is realized only at the time of its execution, at which point the system creates an instance of the same, capturing the state of all of the targets involved in the Workflow. Any changes made to the Components will not influence the execution of WORKFLOW instances already created. The changes are taken into consideration only when a new instance of the WORKFLOW is created by the system. The $\phi-$COMPONENTS serve as the starting point of a WORKFLOW, and because a WORKFLOW can contain multiple $\phi-$COMPONENTS (`w3` in fig. 4.1b), a user makes the choice of the data source for each of those targets when requesting the system to run that WORKFLOW.

On many occasions, scientists want to combine the results of one or more workflows to do further analysis on the aggregated data. Such needs are addressed by COMBINERS. A COMBINER aggregates the output of multiple instances of a single WORKFLOW. Like COMPONENTS, they may define a sequence of high-level programming instructions for the system to execute on the aggregated data. Users may not always want to aggregate the output of all of the instances; therefore, COMBINERS provide a range of options to specify the aggregation period. These include a *start time*—where the data is aggregated from the instances of the stated WORKFLOW on or after the time specified, an *end time*—to make the system consider the instances on or before the mentioned time, or both—to restrict the instances to that period. Other options include the number of past instances to be considered. COMBINERS always define a passive dependency, and they are denoted by a double circle with one solid line and one dashed line (target 2 in fig. 4.1a).

4.3. OUTPUT TYPES

For the long-running spatiotemporal analysis of a continuously updating sensor dataset, frequent access to the intermediate output is critical. Columbus maintains all of the intermediate outputs and supports output types that assist in visualizations at any stage of the workflow. Accordingly, Columbus defines the output types as *Key-Values, Feature, Feature Collection, Multi Collection,* and *Blob* and associates them with targets.

Accessing multidimensional data using indices often creates the problem of mismatched data and makes the instructions in a target difficult to comprehend. Key-Values allow subsequent targets to access this data by means of identifiers, and they enable tabular display. Raw data to

$\phi-$COMPONENTS is always fed in this format. *Feature* and *Feature Collection* output types are based on GeoJSON specification[31], an open standard format for encoding collections of simple geographical features along with their non-spatial attributes using JavaScript Object Notation (JSON). A *Multi Collection* is a sequence of *Feature Collections* that supports the need to output several *Feature Collections* from a single target. Lastly, a *Blob* represents output in any format defined by the end users.

4.4. PIPELINES

A WORKFLOW must be converted into a topological sequence of *pipelines* before slating it for execution. This helps achieve parallelism at the target level while ensuring data locality. Pipelines are derived from the workflow by partitioning the graph at the $\beta-$COMPONENTS while retaining the original dependencies among the targets, generating a series of subgraphs . Each subgraph or *pipeline* is a topological sequence of $\alpha-$COMPONENTS beginning with either a $\phi-$COMPONENT, a $\beta-$COMPONENT, or a COMBINER; accordingly, we refer to these pipelines as $\phi-$, $\beta-$, and $\gamma-$PIPELINES, respectively. Fig 4.2 shows the pipelines for the WORKFLOW `w3` in fig. 4.1b.

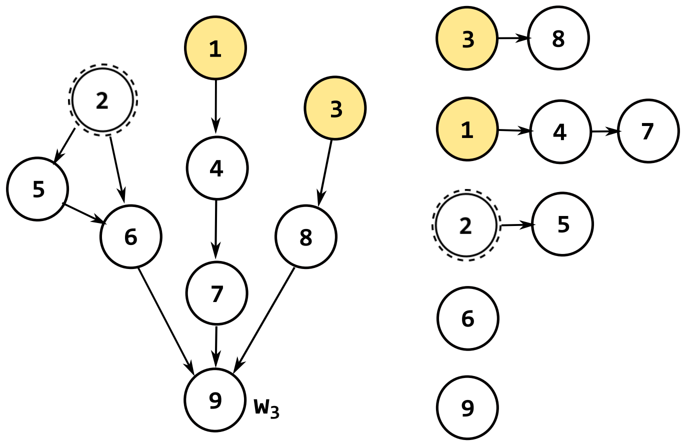


Figure 4.2. A Workflow and its Pipelines

CHAPTER 5

System Architecture

Columbus is built as a multiuser web-based distributed scientific workflow management system that is integrated with the Galileo storage framework and other cloud based storage services for diverse data source choices. It uses a master-worker architecture where the *workers* execute the targets involved in a Workflow while the master manages its execution. In this chapter, we will discuss some critical details of the system with respect to its architectural elements. A high-level architectural block diagram is shown in fig. 5.1.

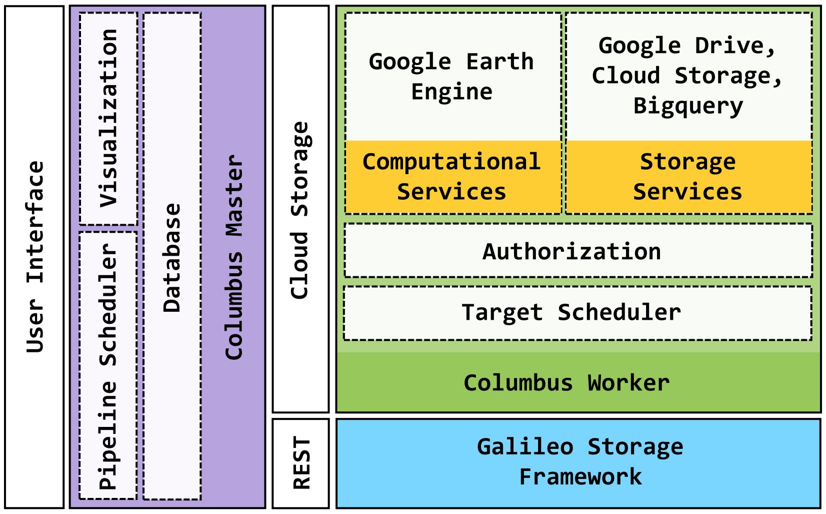


Figure 5.1. Architecture of Columbus

5.1. User Interface

The master provides an authorized web user interface that allows users to create Components, Combiners, and Workflows among others. A Workflow can be run in the system by choosing a data source for every $\alpha-$ Component of that Workflow. Users can make the choice from any of the available cloud storage services or Galileo. The latter allows a user to select the data by space, time, feature, or metadata filters. Columbus creates a visualization of the data stored in Galileo for a chosen dataset that acts as an interface for making spatial queries. Fig. 3.6 illustrates such a visualization and a user query on data stored in the State of Colorado. Furthermore, users can associate priority with the workflows while triggering them so as to make the system prioritize their execution. Columbus provides real-time monitoring of the workflows and allows users to visualize or analyze the results.

5.2. Database

Columbus maintains a database to store users information, their authorizations to cloud services, and their workflow execution traces besides other details. It helps in sharing Workflows among multiple users and aids in visualizations by storing identifiers of Google fusion tables or file paths of target outputs. Storing workflow execution traces in the database generates support for real-time monitoring and provenance. The system makes transparent execution of the workflows a priority; therefore, it hides the underlying distributed environment from the end users. However, administrators can track the activity of workers on demand, in which case the system stores such information in the database and allows the administrators to see the metrics such as workload and resource utilization over time.

5.3. Scheduling

Columbus uses a *dual scheduling* strategy whereby the master runs a *pipeline scheduler* to distribute the pipelines involved in a Workflow, and a worker runs a *target scheduler* to execute the targets in those pipelines. The pipeline scheduler focuses on minimizing the data movement across machines, while the target scheduler ensures fair resource utilization among users and guarantees the successful execution of a target when its resource requirements fall within the system capabilities of the worker. The capacity of a worker indicates the number of targets that could be run in parallel, and it is determined by the container size defined on the master. When a worker connects to the master, it receives configuration parameters such as this, including user authorization credentials. The worker then calculates its capacity based on the container size and reports the same back to the master along with other system information. The master uses this information to update its capacity, $C = \sum_{i=1}^{n} c_i$, where c_i denotes the capacity of a worker i and n is the number of workers. Also, if $0 \leq r_i \leq c_i$ indicates the number of targets running on a worker i, the workload of that worker is estimated as r_i/c_i.

The system defines three scheduling schemes for the execution of Workflows viz. *local*, *remote*, and *hybrid*. The first scheme ensures the highest data locality by allocating targets to the workers housing the data. With the remote scheme, master queries the workers regarding their workload and allocates the targets to the one having the data if its capacity is not full; otherwise, it allocates them to the worker with the lowest workload. Lastly, with the hybrid scheme, workers are requested to send their workload along with the ratio of the number of workflows waiting to those running per user, referred to as the *WR ratio*. The targets are allocated to the worker containing the data regardless of its workload when the WR ratio of the user is less than a preset threshold. Otherwise, this scheme works just like the remote scheme.

Fig. 5.2 represents the communication between master and workers together with the data structures used in the scheduling of workflows. Dashed lines indicate repeated communication, and the $*_{ijk}$ subscript notation indicates the identifier of the User, Workflow, and Target, respectively. Both master and worker scheduler are designed to be iterative processes such that the former maintains a *priority waiting queue* per user, a *ready queue*, and a *backlog* of Workflows. The latter maintains a *ready queue* per Workflow per user, a *shelf*, and a *backlog* of targets, respectively. When users submit Workflows, the pipeline scheduler removes as many Workflows as its capacity C, one from each user in a round-robin fashion, and sorts them on their creation time before queueing them into the ready queue. For each workflow in the ready queue, the master creates an execution plan as a topological sequence of the *pipelines*, distributes the $\phi-$ and $\gamma-$Pipelines to the workers based on the chosen scheduling scheme, and adds the Workflow along with its $\beta-$Pipelines to the *backlog*. The worker, upon receiving the Pipeline, queues the targets it holds into the ready queue of that Workflow for that user. The target scheduler removes as many targets as its remaining capacity from those ready queues, again in a round-robin fashion, triggers their execution as separate processes imposing the memory constraint specified by the container size, and adds them to the *backlog*.

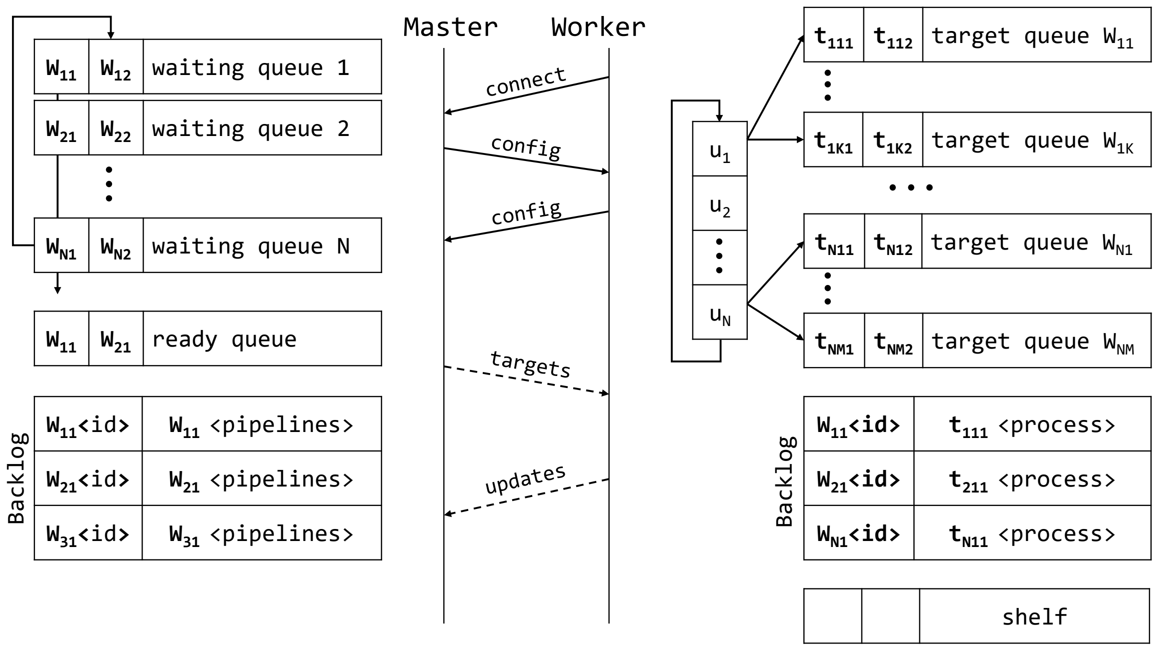


Figure 5.2. Workflow Scheduling

The target processes begin by loading the needed data from the disk of that worker. If the data is not found locally, the same is downloaded from the cloud storage. The target then proceeds with processing user instructions. Currently, GIS computations are pushed to Google Earth Engine [32]. However, domain-specific computations can be done locally if the appropriate software stack is made available to the workers. Once the target processes finish executing the user instructions, they write the output to the local disk and then to the cloud storage. These processes report their success or failure to the worker. If any process fails because of the memory constraint imposed by the worker, the same is added to the *shelf* with double the container size. If the new size is within the capacity of the worker, the target is retried when the assigned size can be allotted. Finally, the target is removed from the *backlog* of the worker, and its status is reported to the master, at which time the pipeline scheduler examines the $\beta-$PIPELINES of that WORKFLOW and slates them for execution if their dependencies were resolved. All of the updates received from the worker get recorded in the database for real-time monitoring and provenance. When all Pipelines of a Workflow finish execution, the Workflow is removed from the *backlog* of the master, yielding room to the next Workflow in the queue.

5.4. Visualizations

Columbus supports both charting and web mapping visualizations for the data associated with a Workflow. The master handles the request made by the user and renders the appropriate visualization. A variety of charting types such as line, bar, spline, area charts, box plots, and heat maps, among others, are included in the system. Fig. 5.3 shows a web-mapping visualization for the output of a scientific workflow. Web-mapping visualizations are enabled through Google Maps API, Google Fusion Tables, and Google Earth Engine.

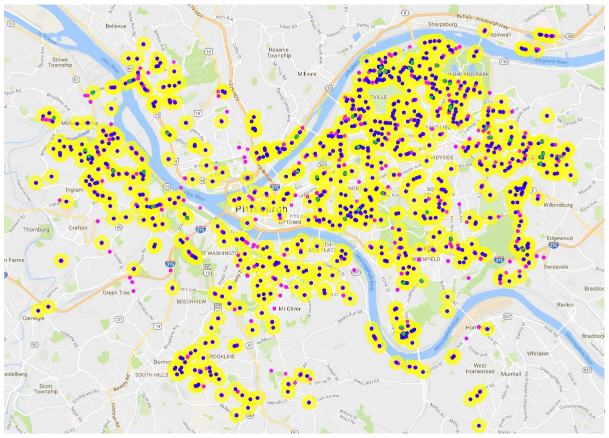


Figure 5.3. Columbus showing the output of a Scientific Workflow.

CHAPTER 6

Evaluation

In this section, we discuss the experiments conducted on our storage system and the distributed workflow engine. Specifically, (1) we compare the performance of our storage system to Geomesa[7] and (2) we evaluate the resource utilization of our system with respect to our scheduling schemes, multiple users, and container size.

6.1. Geomesa

Geomesa is an Apache licensed open source suite of tools that enables large-scale geospatial analytics on cloud and distributed computing systems, letting users manage and analyze the huge spatiotemporal datasets. Geomesa does this by providing spatiotemporal data persistence on top of the Accumulo, HBase, and Cassandra distributed column-oriented databases. It allows rapid access to the data via queries that take full advantage of geographical properties to specify distance and area.

6.2. Experiment Setup

For the purpose of our experiments, we setup a 12-node cluster on the Google cloud platform comprising n1-standard-4 compute engine instances each with 2.6GHz Intel Xeon CPU having 4 vCPUs (threads per core=2, cores per socket=2) and 15 GB of memory. All nodes ran Debian 8 (Jessie) with Linux 3.16.0-4-amd64 kernel and had 512 GB bootable standard persistent disks. The evaluation used Hadoop 2.6.0, Zookeeper 3.4.6, Accumulo 1.7.2, and Geomesa 1.3.0. HDFS block size was defaulted to 128 MB, and the replication level was set to two for Hadoop. Tablet server maps memory was set to 1GB, and index and data cache sizes were set to 512MB and 128MB, respectively, for Accumulo. To run Hadoop NameNode, Accumulo masters, and Zookeeper server, another Google compute engine instance with the same configuration was used. All nodes had the limit on number of open files set to 65536 and ran Oracle JDK Java Runtime version 1.8.0_121-b13.

We used the natural gas leak mobile sensor dataset collected as part of a project funded by the Environment Defense Fund that covers multiple regions and divisions of the United States. The dataset was collected over 4 years and consists of more than 25 million observations having 40 dimensions.

6.3. Query Performance

To evaluate the performance of the queries involving space or time, we considered ten different queries in each of the six categories listed in table 6.1.

Table 6.1. Query categories and number of resulting observations

Query	Number of resulting observations(K) / Size(MB)		
Category	Lowest	Mean	Highest
Spatial Region	603.6 / 196.5	1130 / 392.8	1655 / 607.9
Temporal Month	585.8 / 201.5	819.9 / 280.2	1097 / 372.5
Temporal Day	75.69 / 25.89	78.86 / 26.98	83.92 / 29.12
Spatial Year	28.35 / 9.590	390.9 / 134.5	728.6 / 253.9
Spatial Month	136.2 / 46.26	273.9 / 95.02	454.5 / 161.8
Spatial Day	21.67 / 7.270	31.98 / 11.06	43.83 / 14.89

Spatial region in any query category spans a four character Geohash region covering an area of approximately 24x12 square miles (39km x 19.5km). Temporal month and day categories include queries made for one day and one month, respectively. Spatial year, month, and day categories include queries on both space and time periods of lengths one day, one month, and one year. Galileo is capable of addressing the spatial queries efficiently by avoiding the processing of a block when it lies inside the spatial query. Therefore, to show that the response times are not the best-case values of Galileo and to demonstrate the influence of the Geohash precision used for storing the blocks on the response time, we considered storing the same dataset in Galileo five times, each having a different storage precision or network organization but temporally distributed on day. We ran the same queries on all five datasets and compared the response times to that of Geomesa. Each dataset of Galileo is identified in the plot by the values of p and n that denote the number of characters in Geohash used for storing the blocks and the number of nodes per group in the network organization of Galileo, respectively. Each query was run five times, and the average response time is reported. The circular points in the boxplot are the mean processing times of the actual queries, and the triangular points are outliers.

For a fixed number of nodes in the cluster, an increase in the value of n decreases the parallelism on spatial queries and increases the same for temporal queries. And for any network organization and a given spatial query, an increase in the value of p results in a greater number of blocks and helps Galileo avoid processing some of them, while a decrease in the same results in fewer blocks and mandates their processing. The results included in fig. 6.1 and 6.2 capture this behavior.

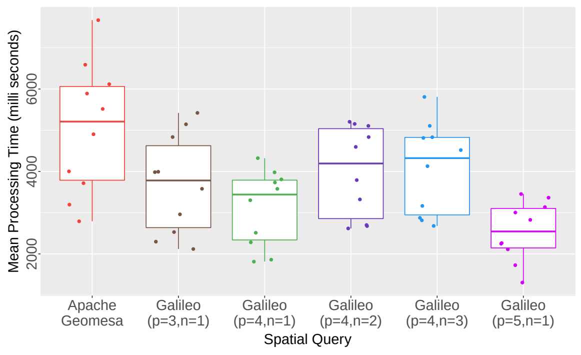
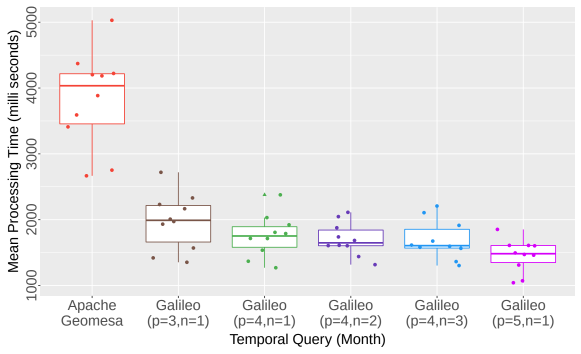
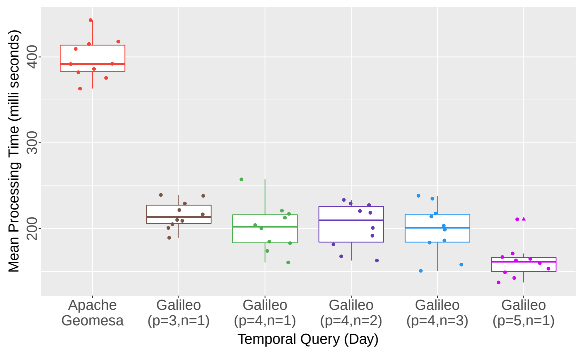


Figure 6.1. Spatial Query Performance



(a) Temporal Month



(b) Temporal Day

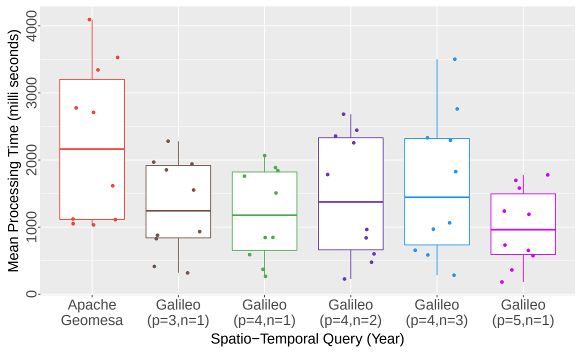
Figure 6.2. Temporal Query Performance

Results show that Galileo performs better than Geomesa in five out of six query categories regardless of the storage precision and network organization used for the dataset. Temporal year category was not considered because the performance of Geomesa was degrading linearly with the rise in number of results (fig. 6.2a and 6.2b). For spatiotemporal queries on day, Galileo is up to 2 times slower than Geomesa. This is because Galileo writes the results to the disk before sending them to the client. Moreover, when it comes to spatial queries on day, paralellism is limited to a single node if the data was distributed on day. To improve the performance in such cases, blocks must be stored in Galileo for smaller spatial area than spatial queries of interest. This can be seen in fig. 6.3c for the dataset (p=5, n=1), which performs closer to Geomesa than other datasets in Galileo. Galileo can perform better if the number of results increases, as observed from figs. 6.3a and 6.3b. Table 6.1 also lists the cumulative result file size on all of the nodes in the cluster for each query category. It should be noted that the file size is not dependent on the number of resulting observations alone but rather on the values of the dimensions of those observations.

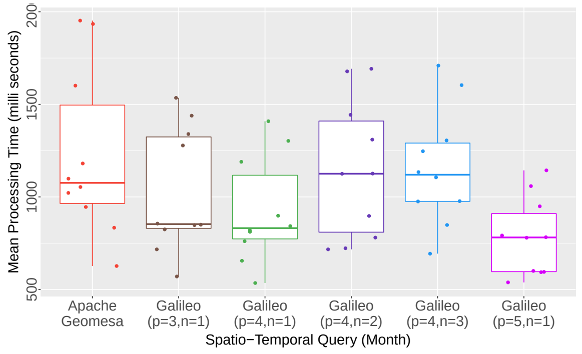
6.4. Resource Utilization

To evaluate the effectiveness of our scheduling strategy and distributed execution, we ran Columbus workers alongside Galileo on the same 12-node cluster and deployed the Columbus master on another Google compute engine instance of type n1-highcpu-8 having 8vCPUs, 7.2GB of memory, and 256GB bootable standard persistent disk. Other configuration details are the same as discussed before. Because Columbus was written in Python and Galileo in Java, Columbus master interacts with Galileo through a RESTful web service interface. In each node of the cluster, we reserved 4GB of memory for Galileo and used the rest for Columbus worker with container size set to 1024MB.

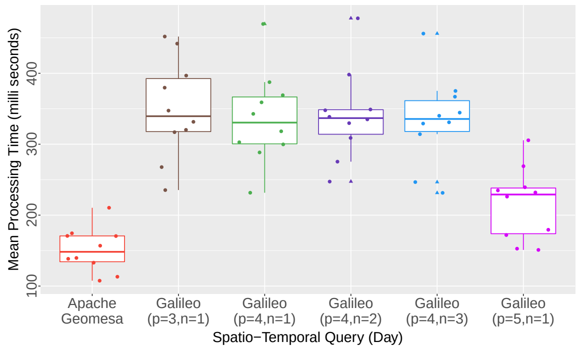
We ran a scientific workflow as a single user on one year of data in all of the spatial regions of the dataset (p=4, n=1) stored in Galileo. We triggered the run, asking Columbus to process the workflow per day, and it resulted in a total of 773 workflow instances. This is more than the



(a) Spatial Year



(b) Spatial Month



(c) Spatial Day

Figure 6.3. Spatiotemporal Query Performance

number of days in that year because some days had more than one block stored in Galileo. We repeated the execution of these instances once for each scheduling scheme and measured the cluster utilization as a percentage of the containers occupied. Fig 6.4 compares the cluster utilization and time taken to finish the execution of all of the instances. As the data stored was distributed temporally on day, some nodes had to process more workflows than others when using the *local* scheduling scheme, which resulted in lower cluster utilization and higher time to finish, as shown in the figure. The *remote* scheduling scheme had the lowest time to finish and better utilization than the *local* scheme. However, it took slightly more time to finish the execution than the *hybrid* scheme for some of the instances at around 500 to 625 seconds time frame. This is because of the data transfer to the remote machine.

To demonstrate the reliability or successful execution of workflows when they are within the resource capabilities of a worker, we reduced the container size to 512MB and repeated the same set of workflow instances as before using both *remote* and *hybrid* scheduling schemes. Fig. 6.5 compares the utilization and time for the two schemes with reduced container size to the original *hybrid* scheme with 1024MB container size.

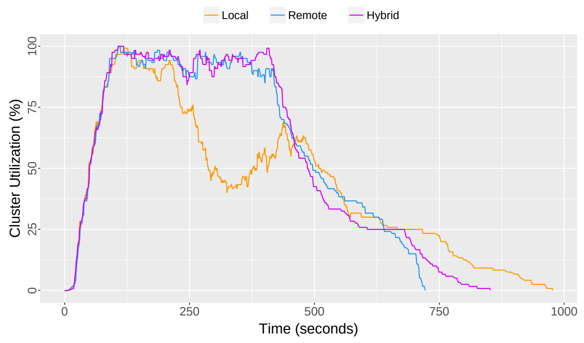


Figure 6.4. Cluster utilization for different scheduling schemes

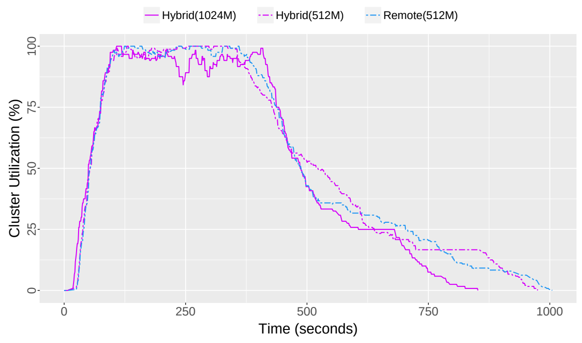
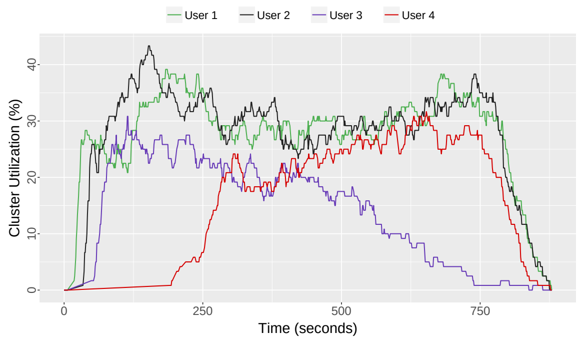


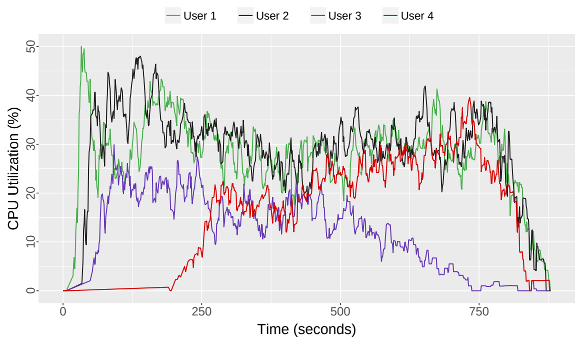
Figure 6.5. Reliable workflow execution with reduced container size

It can be observed that the new container size resulted in higher cluster utilization and time to finish than the previous case. This is because several instances failed due to insufficient memory, and they were retried with double the container size. Moreover, the higher cluster utilization was because of the fact that the containers were reserved for the failed instances. It should be noted that the *remote* scheduling scheme with smaller container had higher time to finish because a failure in execution will result in retrying and repeated data transfer. The takeaway from this experiment is that the *hybrid* scheduling scheme with appropriate WR ratio will yield better utilization and time to finish by ensuring fair data locality.

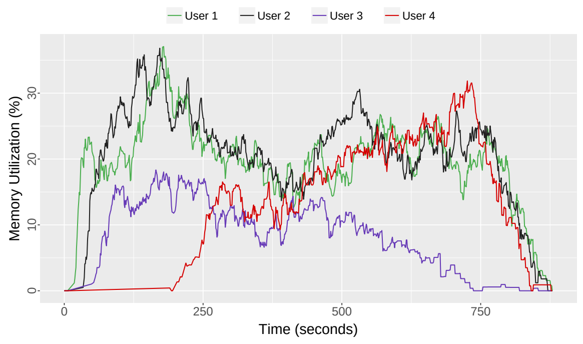
Finally, we conducted an experiment to demonstrate how resources were shared among multiple users by running the workflow instances for four users. We triggered the same instances for users one and two, and a different set of instances for user three. The instances were created for users one after the other, in the order of their numbering. We triggered yet another set of instances for user four at around three minutes after the execution started for other users. Fig. 6.6 shows the resource utilization graphs for these four users in terms of cluster, CPU, and memory utilization. The graphs clearly show that the resources are fairly shared among the users regardless of when the submission of instances took place. However, users who created the instances first will have their execution begin early.



(a) Cluster Utilization



(b) CPU Utilization



(c) Memory Utilization

Figure 6.6. Resource allocation among four users using hybrid scheduling scheme having WR ratio as one.

CHAPTER 7

Conclusion and Future Work

In this work, we presented a cloud-based, multiuser, distributed workflow engine that enables efficient composition, scheduling, and execution of scientific analysis workflows over voluminous spatiotemporal sensor datasets.

RQ1. To enable composing data-bounded workflows, target-oriented workflow modelling allows users to specify dependencies such as data availability and execution of upstream components. This allows the scheduler to delay allocation of the physical resources until the data is available and prevents undesired resource idling.

RQ2. Columbus extends Galileo's hierarchical DHT indexing scheme with customizable temporal granularities. Since temporal data proximity is preserved, Columbus outperforms Geomesa for most of the temporal and spatial data retrieval benchmarks as presented in chapter 6.

RQ3. Columbus schedules and executes tasks in data-bounded workflows with hierarchical queues and three different data locality scheduling schemes. This multipronged approach was shown to be effective in utilizing computing resources and providing fair scheduling to multiple users.

While the proposed system is in good shape for spatiotemporal analytics on data at scale, there is room for improvements to the system. This involves (1) changing the scheduling algorithm to immediately execute the workflows of new users who submit them when the cluster utilization is at its maximum by killing the processes of over-utilizing users and adding them back to their queues, (2) improving spatial queries to allow multi-geometries, (3) providing support for other programming languages in specifying the computation for targets, (4) exporting workflows as Columbus archive files to share with external users, (5) root cause analysis, and (6) improved security for executing end user computations, besides others. We leave such improvements to the system for future work.

References

[1] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[2] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[3] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome biology*, vol. 11, no. 8, p. 1, 2010.

[4] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015. Funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575.

[5] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. Truong, A. Villazon, and M. Wieczorek, "Askalon: A development and grid computing environment for scientific workflows," in *Workflows for e-Science*, pp. 450–471, 2007.

[6] M. Malensek, S. L. Pallickara, and S. Pallickara, "Galileo: A framework for distributed storage of high-throughput data streams," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pp. 17–24, IEEE, 2011.

[7] A. Fox, C. Eichelberger, J. Hughes, and S. Lyon, "Spatio-temporal indexing in non-relational distributed databases," in *Big Data, 2013 IEEE International Conference on*, pp. 291–299, IEEE, 2013.

[8] D. Talia, “Workflow systems for science: Concepts and tools,” *ISRN Software Engineering*, no. 404525, p. 15, 2013.

[9] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, “A survey of data-intensive scientific workflow management,” *Journal of Grid Computing*, vol. 13, no. 4, pp. 457–493, 2015.

[10] V. Curcin and M. Ghanem, “Scientific workflow systems-can one size fit all?,” in *2008 Cairo International Biomedical Engineering Conference*, pp. 1–9, IEEE, 2008.

[11] D. De Roure, C. Goble, and R. Stevens, “The design and realisation of the virtual research environment for social sharing of workflows,” *Future Generation Computer Systems*, vol. 25, no. 5, pp. 561–567, 2009.

[12] I. Taylor, M. Shields, I. Wang, and A. Harrison, “The triana workflow environment: Architecture and applications,” in *Workflows for e-Science*, pp. 320–339, 2007.

[13] B. Liu, B. Sotomayor, R. Madduri, K. Chard, and I. Foster, “Deploying bioinformatics workflows on clouds with galaxy and globus provision,” in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pp. 1087–1095, IEEE, 2012.

[14] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor, “Galaxy cloudman: delivering cloud compute clusters,” *BMC Bioinformatics*, vol. 11, no. 12, p. S4, 2010.

[15] I. Foster and C. Kesselman, “Globus: A metacomputing infrastructure toolkit,” *Int. J. High Perform. Comput. Appl.*, vol. 11, pp. 115–128, June 1997.

[16] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, “Swift: Fast, reliable, loosely coupled parallel computation,” *IEEE Int. Conf. on Services Computing - Workshops (SCW)*, pp. 199–206, 2007.

[17] F. Marozzo, D. Talia, and P. Trunfio, “Js4cloud: script-based workflow programming for scalable data analysis on cloud platforms,” *Concurrency and Computation: Practice and Experience*, vol. 27, pp. 5214–5237, 2015.

[18] I. Raicu, *Many-task Computing: Bridging the Gap Between High-throughput Computing and High-performance Computing.* ProQuest / UMI, 2011.

[19] E. Ogasawara, D. de Oliveira, F. Chirigati, C. E. Barbosa, R. Elias, V. Braganholo, A. Coutinho, and M. Mattoso, “Exploring many task computing in scientific workflows,” in *Proceedings of the 2Nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS ’09, (New York, NY, USA), pp. 2:1–2:10, ACM, 2009.

[20] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, “Falkon: A fast and light-weight task execution framework,” in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, SC ’07, (New York, NY, USA), pp. 43:1–43:12, ACM, 2007.

[21] S. Lee Pallickara and M. Pierce, “Swarm: Scheduling large-scale jobs over the loosely-coupled hpc clusters,” *2008 IEEE Fourth International Conference on eScience*, vol. 00, pp. 285–292, 2008.

[22] Z. Zhang, D. S. Katz, M. Ripeanu, M. Wilde, and I. T. Foster, “Ame: An anyscale many-task computing engine,” in *Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science*, WORKS ’11, (New York, NY, USA), pp. 137–146, ACM, 2011.

[23] K. Wang, K. Qiao, I. Sadooghi, X. Zhou, T. Li, M. Lang, and I. Raicu, “Load-balanced and locality-aware scheduling for data-intensive workloads at extreme scales,” *Concurrency and Computation: Practice and Experience*, vol. 28, no. 1, pp. 70–94, 2016. CPE-14-0369.R2.

[24] L. Singh and S. Singh, “Article: A survey of workflow scheduling algorithms and research issues,” *International Journal of Computer Applications*, vol. 74, pp. 21–28, July 2013. Full text available.

[25] M. Malensek, S. Pallickara, and S. Pallickara, “Polygon-based query evaluation over geospatial data using distributed hash tables,” in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pp. 219–226, IEEE Computer Society, 2013.

[26] M. Malensek, S. Pallickara, and S. Pallickara, “Geometry and proximity constrained query evaluations over large geospatial datasets using distributed hash tables,” *Computing in Science & Engineering*, no. 1, pp. 1–1, 2014.

[27] M. Malensek, S. Pallickara, and S. Pallickara, "Analytic queries over geospatial time-series data using distributed hash tables," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1408–1422, 2016.

[28] G. Niemeyer, "Geohash." Online: http://www.geohash.org/, Accessed: 2016-11-06.

[29] National Oceanic and Atmospheric Administration, NOAA, "National centers for environmental information." Online: http://www.ncdc.noaa.gov/societal-impacts/wind/, Accessed:2016-11-06.

[30] T. Mitchell, "Web mapping illustrated: using open source gis toolkits," *O'Reilly Media, Inc.*, 2005.

[31] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, and C. Schmidt, "The geojson format specification." Online: http://geojson.org/geojson-spec.html, Accessed: 2016-11-06.

[32] Google Earth Engine Team, "Google earth engine: A planetary-scale geo-spatial analysis platform," 12 2015. Online: https://earthengine.google.com, Accessed: 2016-11-06.

[33] Attribution for Calendar Icon[1] and Earth Icon[2] in figures 3.1 and 3.2

[1]Made by Freepik (http://www.freepik.com) from Flaticon (http://www.flaticon.com) is licensed by Creative Commons BY 3.0 (http://creativecommons.org/licenses/by/3.0/)

[2]Made by Flat Icons (http://www.flaticon.com/authors/flat-icons) from Flaticon (http://www.flaticon.com) is licensed by Creative Commons BY 3.0 (http://creativecommons.org/licenses/by/3.0/)