Columbus: Enabling Scalable Scientific Workflows for Fast Evolving Spatio-Temporal Sensor Data

Johnson Charles Kachikaran Arulswamy
Dept. of Computer Science
Colorado State University
Fort Collins, USA
jcharles@cs.colostate.edu

Sangmi Lee Pallickara
Dept. of Computer Science
Colorado State University
Fort Collins, USA
sangmi@cs.colostate.edu

***Abstract*—As spatiotemporal sensor data collections are becoming more available, the ability to generate insights from them is turning increasingly complex for scientists in this domain. Organizing such voluminous data and keeping track of analysis results get tedious from day to day. In this paper, we propose a system comprising a distributed workflow execution engine, a storage framework with flexible spatiotemporal data dispersion and indexing scheme for fast data search and retrieval, and a scheduling scheme that aims at minimizing data movement across machines while simultaneously ensuring fair and efficient allocation of resources among multiple users. Our empirical evaluations demonstrate the feasibility of our approach and its ability to scale.**

***Keywords*-scientific workflow; spatiotemporal; distributed system; workflow scheduling; storage framework;**

I. 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) computations that are long running, network I/O bound, and executed based on data availability on unbounded data streams, (2) voluminous data that arrive at high rates, (3) data selection constrained by arbitrary space and time, and (4) volatile resource requirements. In this paper, we define *data-bounded workflow* as a workflow that accounts for characteristics of data generated in sensing environment with stages accounting for availability of data, their generation rates, and the accompanying data volumes. Furthermore, in a data-bounded workflow, inputs or intermediate data may not be available immediately; however, polling is not a viable option. To execute data-bounded workflow, the workflow engine should allow users to compose workflows that operate on data from sources that are not currently online or are dormant. The workflow must process data streams in real-time in a shared cluster without the need to preempt computing resources. The crux of this paper is to support the composition and orchestration of data-bounded workflow in the context of fast-evolving voluminous real-time datasets generated in sensing environments.

A. Scientific Challenges

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

1) **Voluminous data**: As the number of sensors grows, cumulative data generated by the sensor network increases dramatically.
2) **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.

3) **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.
4) **Scalability**: The proposed approach must scale with increases in the number of workflows and data volumes.

B. 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 we explore in this paper 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. (§ III)

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 allow the workflow engine to allocate tasks with high data locality. (§ II)

RQ-3 How can we schedule and execute data-bounded workflows effectively in high-throughput analysis environment? 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. (§ IV)

C. Overview of our Approach

Columbus is a distributed workflow engine 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 orchestration of computations. To facilitate the composition of complex data-bounded workflows, we propose a target-oriented workflow model that not only address 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 worker 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 number of workflows waiting to running per user.

D. Paper 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 long-running time-series 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 25 million observations, each of which contains 40 attributes for a specific location. Our benchmarks also contrast the data retrieval performance of Columbus with Geomesa [7].

E. Paper Organization

The rest of the paper is organized as follows. Section § II outlines our data dispersion scheme and query processing. § III describes the design details of the Columbus system while § IV covers the architectural details and workflow scheduling. Performance evaluation is included in § V followed by a survey of related work in § VI. Finally, § VII concludes the paper and describes our future research directions.

II. Storage Framework

For our storage requirements, we extended Galileo [6], [8]–[10], 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.

A. 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 destination [6]. In the original partitioning scheme of Galileo, a group of nodes was determined by Geohash [11], whose length indicated its precision, and a node within the group was identified by SHA-1 hash. In this paper, 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.

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 National Oceanic and Atmospheric Administration every hour, but the climatological mean wind speed is reported every month [12]. 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.

B. The Geoavailability Grid

A *geoavailability grid* is a spatial indexing data structure [8] that translates points in space to a reduced-resolution coordinate system for indexing purposes. It is described by a bitmap denoted by 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 and get the raw data points present in a block. Polygonal queries are decomposed into smaller polygons that cover the area of the Geohash, which the grid represents. We create a 30-bit resolution grid on demand for the necessary blocks in each node. If the blocks are stored at a precision of four characters, then the actual resolution of the grid can be seen as 50-bit which gives accuracy to nearly one meter in space. Fig. 3 shows a few geoavailability grids for the query made in fig. 1.

C. Query Evaluation

Galileo supports a variety of queries including feature queries, range queries on space or time, 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 on the dimensions of the dataset or metadata of the blocks, simply *filters*.

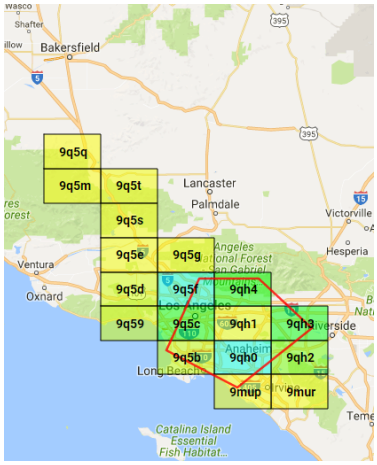
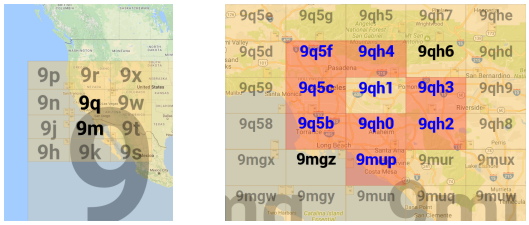


Fig. 1: Columbus showing the data reported by Galileo in the state of California and a spatial query made around Los Angeles, CA

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.



(a) Node Level (b) Block Level

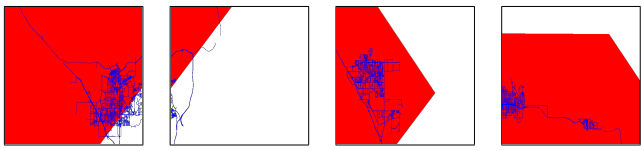
Fig. 2: Determining intersecting Geohashes

Fig. 1 shows a spatial query made in the region of California state. 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 (example, one in 9q5f) and constructs the Geohash value as 9q. It computes all the eight neighbors of this Geohash as shown in fig. 2a. The intersecting Geohashes are highlighted in bold. Of the neighbors, only 9m is found to be intersecting, and the system recomputes its neighboring Geohashes by excluding the ones already processed. The resulting set is used to identify the set of destination nodes.

At the destination nodes, the intersecting Geohashes are obtained using a *recursive binary partitioning* technique on each Geohash region present at the destination that is known to intersect the spatial query. A query is then evaluated on the metadata graph for the set of intersecting Geohashes to obtain the blocks. If the request is made for the raw data, then for each block that is intersecting the polygon, data is loaded into a geoavailability grid constructed with the same precision as that of the block and results are obtained by matching the bitmap representing the grid against that of the polygon. For the blocks that are enclosed with in the polygon, all the raw data in the block is returned when there is no filter. If a filter is specified within the query, the filtering is done by loading the data into a temporary hierarchical graph and the filtered results are loaded into the grid. Fig. 2b shows the intersecting Geohashes in bold and the blocks found for those hashes in blue. The shaded Geohashes indicate the blocks that are loaded into the geoavailability grid for further processing. Fig. 3 shows a few grids and the polygon for these blocks at the node having the 9q region.

Temporal queries made in accordance with the temporal type specified for the temporal partitioning scheme are redirected to the nodes involved in the matching group. Otherwise, they are treated as regular feature queries and are directed to all

the nodes in the system. These queries are addressed by the metadata graph at the respective destinations. Queries made on both space and time are resolved first for time followed by space, to obtain the set of destination nodes and are addressed in the same order at the destinations.

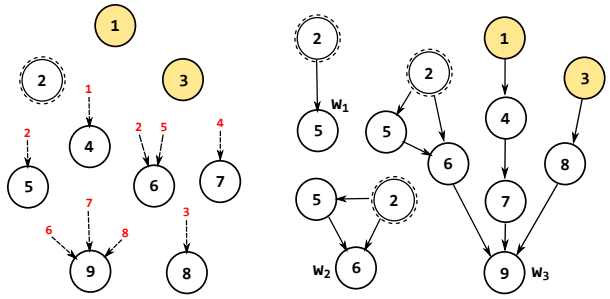


(a) 9qh0 (b) 9qh2 (c) 9qh3 (d) 9qh4

Fig. 3: A few Geoavailability grids for the query in fig.1 at nodes having 9q Geohash

III. System Design

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



(a) Targets (b) Workflows

Fig. 4: Modeling Workflows from Targets

A. Target-Oriented Workflow Model

Designing a long-running 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 inter-dependent 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* [13]. 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. 4a captures a few targets defined in the system, and some workflows composed from those targets are shown in fig. 4b. 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.

B. 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, $\alpha, \beta-$Components are denoted by circles with a solid line and $\phi-$Components are denoted by filled circles. Fig. 4a shows eight Components, of which 1 and 3 are $\phi-$Components, 4, 5, 7, and 8 are $\alpha-$Components, and 6, 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 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 (w_3 in fig. 4b), 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 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 to 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. 4a).

C. Output Types

For the long-running spatiotemporal analysis of continuously updating sensor dataset, frequent access to the intermediate output is critical. Columbus maintains all 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 multi-dimensional 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 [14], 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.

D. 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 also 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, $\beta-$COMPONENT or a COMBINER; accordingly, we refer to these pipelines as $\phi, \beta, \gamma-$PIPELINES respectively. Fig 5 shows the pipelines for the workflow w_3 in fig. 4b.

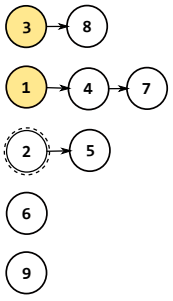


Fig. 5: Pipelines of W_3

IV. SYSTEM ARCHITECTURE

Columbus is built as a multi-user web-based distributed scientific workflow management system that is tightly coupled with the Galileo storage framework. It is also integrated with 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. The system aims at transparent execution of the workflows and therefore hides the underlying distributed environment from the end users.

Like many other workflow engines, it maintains a database to store security information such as users authorizations to cloud services, workflow execution traces to provide real-time monitoring and support for provenance, and to allow sharing of workflows with other users. It uses a cloud storage service for fault tolerance and data movement among workers and or master. This helps in avoiding communication with other workers during and after the execution of workflows. Fig. 6 shows a high-level architectural block diagram of the system.

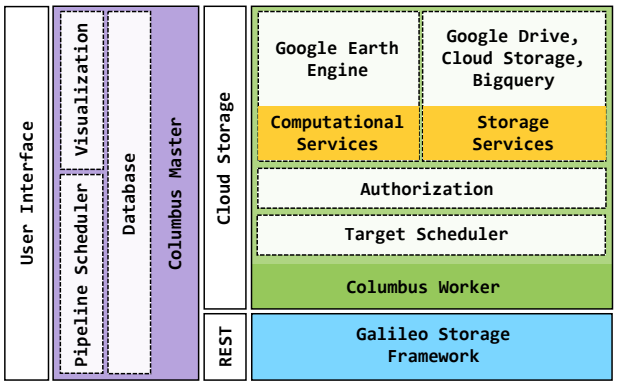


Fig. 6: Architecture of Columbus

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 allocation 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, then 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 7 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 schedulers 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 master 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 $\phi, \gamma-$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*.

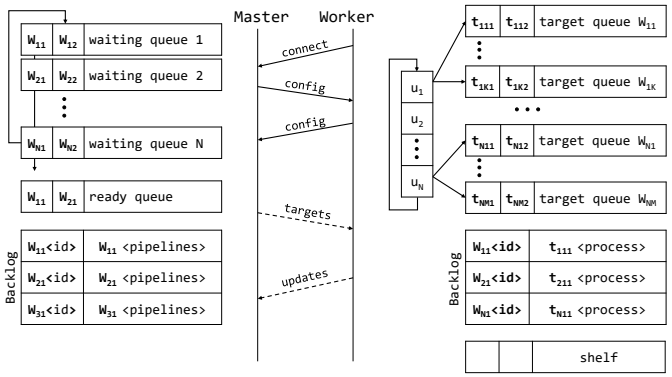


Fig. 7: Workflow Scheduling

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 [15]. 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 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.

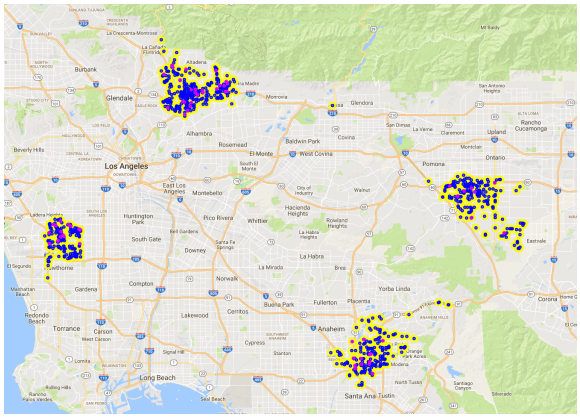


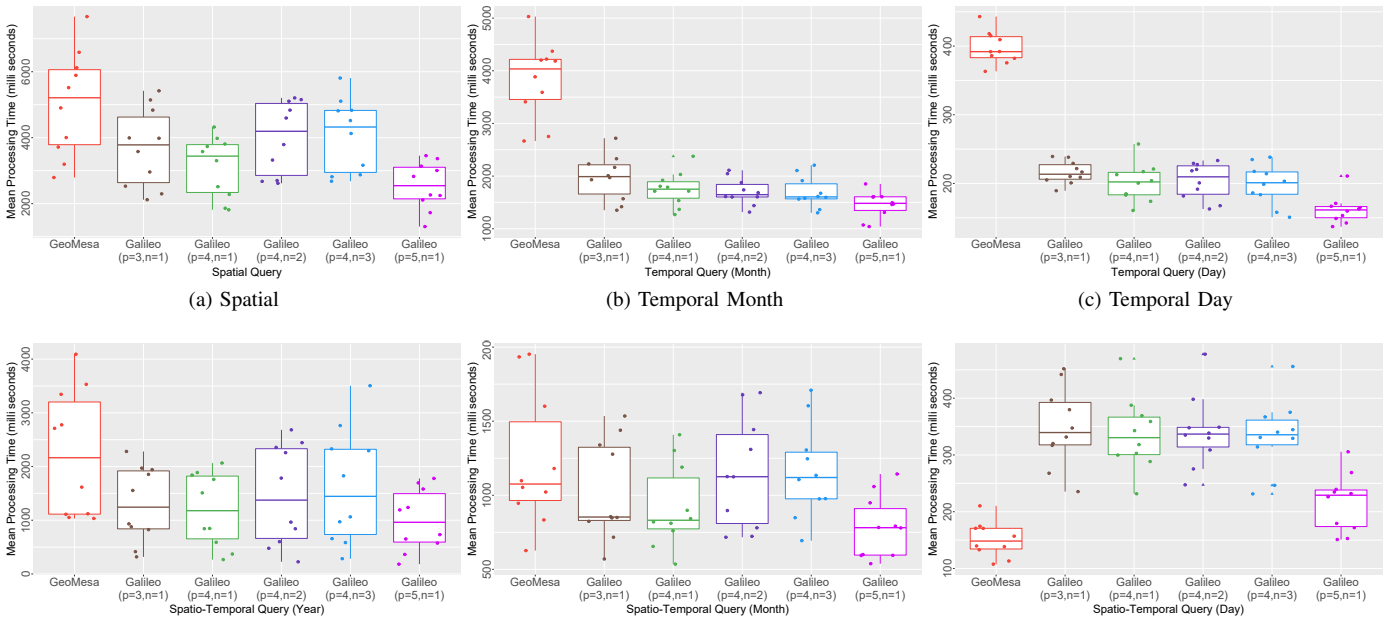
Fig. 8: Columbus showing the output of a Scientific Workflow.

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, and area charts, box plots, and heat maps among others, are included in the system. Fig. 8 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.

V. 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], another spatiotemporal storage framework for data at scale built on top of other big data systems, and (2) we evaluate the resource utilization of our system with respect to our scheduling schemes, multiple users, and container size.

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 Apache Hadoop 2.6.0, Apache Zookeeper 3.4.6, Apache 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



(a) Spatial (b) Temporal Month (c) Temporal Day

(d) Spatial Year (e) Spatial Month (f) Spatial Day

Fig. 9: Comparison of query processing time averaged over five runs for ten different queries in each of the six categories

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 Environmental 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.

A. *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 I. 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 as shown in fig. 9. 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 ones 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 in fig. 9 capture this behavior.

Table I: 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

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. 9b and 9c). 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, parallelism 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. 9f for the dataset (p=5, n=1) which performs close to Geomesa than other datasets in Galileo. Galileo can perform better if the number of results increases, as observed from the fig. 9d and 9e. Table I also lists the cumulative result file size on all 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.

B. 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.

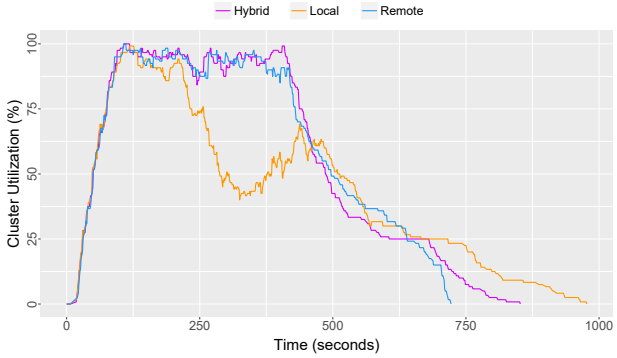


Fig. 10: Cluster utilization for different scheduling strategies

We ran a scientific workflow as a single user on one year of data in all 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 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 10 compares the cluster utilization and time taken to finish the execution of all 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 *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. 11 compares the utilization and time for the two schemes with reduced container size to the original *hybrid* scheme with 1024MB container size.

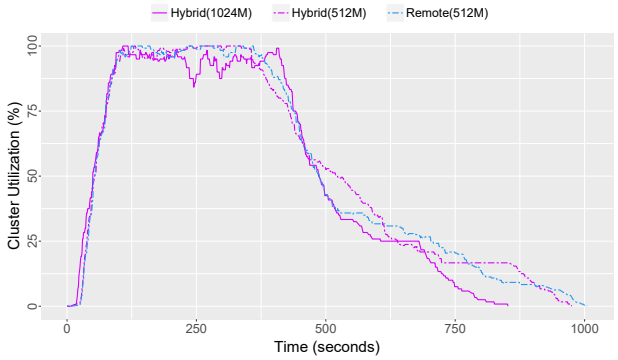


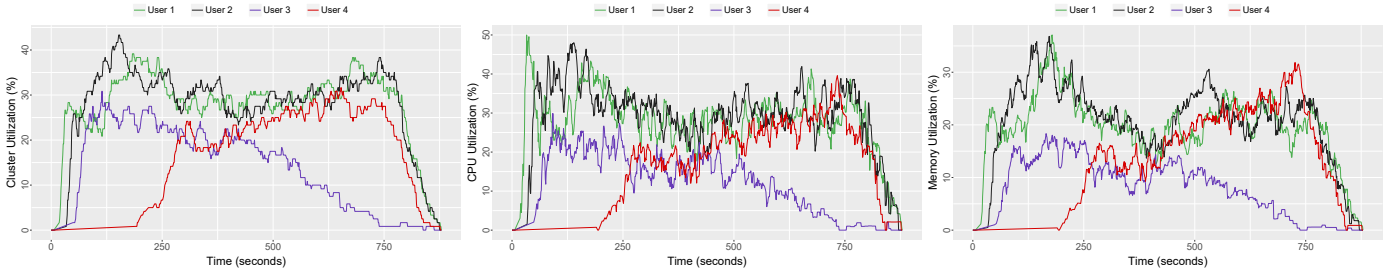
Fig. 11: Reliable workflow execution with reduced container size

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. 12 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.

VI. Related Work

Several scientific workflow management systems exist [16], [17] 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,



(a) Cluster Utilization (b) CPU Utilization (c) Memory Utilization

Fig. 12: Resource utilization among four users using hybrid scheduling strategy having WR ratio as one.

biology, computational engineering, and earth sciences. Yet, new systems emerge from time to time because it is unlikely for any single system to handle such diverse domain needs [18].

Kepler [1], Taverna [2], and Triana [19] are open-source domain-independent scientific workflow management systems based on Java. These systems allow users to create workflows through visual formalisms such as dragging and dropping components into the canvas and allow them to specify the dependencies among the components either as a directed graph or a directed acyclic graph. They leverage the power of distributed computing through web and grid services for parallel processing of the workflows. Research has shown that the composition of such services can also be automated using Artificial Intelligence [20]. Kepler and Taverna also allow sharing of the workflows through myExperiment [21], a collaborative environment where scientists can safely publish their workflows and *in silico* experiments. Galaxy [3] is another scientific workflow system that allows users to import workflows from myExperiment but was particularly developed for data-intensive biomedical research. However, it uses Gridway to execute tasks in the grid and can exploit Globus [22] and CloudMan [23] to achieve dynamic computing and storage provisioning across computing nodes.

Unlike the previous systems that provide the application developers with a non-transparent grid, the Pegasus project [4] and Askalon project [5] aim at shielding the user from the details of the underlying execution environment or the particulars of the low-level specifications required by the middleware. Askalon allows a user to compose grid workflow applications graphically using a UML-based workflow composition and modeling service. It provides an XML-based programming interface to the application developers to represent the workflow that can be submitted to Askalon run-time system for scheduling and reliable execution on Grid infrastructures. On the other hand, Pegasus takes an abstract workflow represented as a directed acyclic graph in an XML file and transforms it into an executable through a series of refinements. The abstract workflow would be independent of resources and the goal of Pegasus is to find a good mapping of the tasks to the available resources necessary for execution. Both Askalon and Pegasus make use of the Globus Toolkit [24] for resource discovery and job submission.

Script-based workflows such as Swift [25] and JS4Cloud [26] exist as an alternative to 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 which 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 allows users to specify long-running workflows having volatile resource requirements, even before the data is available.

Researchers have also explored many-task computing (MTC) [27] in scientific workflows [28]. 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 [29], SWARM [30], and AME [31] 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 [32]. A survey of the workflow scheduling algorithms [33] 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 proposed scheduling mechanism also ensures reliable workflow execution given the fact that the memory requirements of the tasks involved in a data-bounded workflow are volatile.

VII. Conclusion

In this paper, we presented a cloud-based, multi-user, 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 modeling 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 § V. **RQ3.** Columbus schedules and executes tasks in data-bounded workflows with hierarchical queues and three different data locality scheduling schemes. This multi-pronged approach was shown to be effective in utilizing computing resources and providing fair scheduling to multiple users.

ACKNOWLEDGMENT

This research was supported by funding from the US National Science Foundation's Advanced Cyberinfrastructure (ACI-1553685), the Environmental Defense Fund (0164-000000-10410-100), and Geo for Google Cloud Credits Beta Program.

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. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/20/17/3045.abstract

[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. [Online]. Available: http://pegasus.isi.edu/publications/2014/2014-fgcs-deelman.pdf

[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*, 2007, pp. 450–471.

[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*. IEEE, 2011, pp. 17–24.

[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*. IEEE, 2013, pp. 291–299.

[8] 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*. IEEE Computer Society, 2013, pp. 219–226.

[9] ——, "Geometry and proximity constrained query evaluations over large geospatial datasets using distributed hash tables," *Computing in Science & Engineering*, no. 1, pp. 1–1, 2014.

[10] ——, "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.

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

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

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

[14] 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.

[15] 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.

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

[17] 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.

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

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

[20] S.-C. Oh, D. Lee, and S. R. Kumara, "Effective web service composition in diverse and large-scale service networks," *IEEE Transactions on Services Computing*, vol. 1, no. 1, pp. 15–32, 2008.

[21] 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.

[22] 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:*. IEEE, 2012, pp. 1087–1095.

[23] 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. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-11-S12-S4

[24] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *Int. J. High Perform. Comput. Appl.*, vol. 11, no. 2, pp. 115–128, Jun. 1997. [Online]. Available: http://dx.doi.org/10.1177/109434209701100205

[25] 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.

[26] 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.

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

[28] 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*, ser. MTAGS '09. New York, NY, USA: ACM, 2009, pp. 2:1–2:10. [Online]. Available: http://doi.acm.org/10.1145/1646468.1646470

[29] 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*, ser. SC '07. New York, NY, USA: ACM, 2007, pp. 43:1–43:12. [Online]. Available: http://doi.acm.org/10.1145/1362622.1362680

[30] 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.

[31] 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*, ser. WORKS '11. New York, NY, USA: ACM, 2011, pp. 137–146. [Online]. Available: http://doi.acm.org/10.1145/2110497.2110513

[32] 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. [Online]. Available: http://dx.doi.org/10.1002/cpe.3617

[33] L. Singh and S. Singh, "Article: A survey of workflow scheduling algorithms and research issues," *International Journal of Computer Applications*, vol. 74, no. 15, pp. 21–28, July 2013, full text available.