Aeromancer: A Workflow Manager for Large-Scale MapReduce-Based Scientific Workflows

Nabeel Mohamed, Nabanita Maji, Jing Zhang, Nataliya Timoshevskaya and Wu-chun Feng
Department of Computer Science
Virginia Tech
Email: {nabeel, nabanita, zjing14, timnatvt, wfeng}@vt.edu

***Abstract*—The Hadoop framework has gained significant attention from the scientific community due to its applicability to large-scale data analysis in many areas. This analysis often involves multiple stages of processing, which in turn, constitutes a workflow. While some stages of a workflow are mandatory, others are subject to the type of analysis to be done. In addition, a workflow may possess data dependencies between stages that must be enforced, and it may exhibit varying levels of sensitivity. The resources needed for such data analysis can range from a laptop to in-house clusters (or private cloud) to a public cloud. Managing such workflows, while using such a gamut of computing resources, is an unnecessarily arduous task for domain scientists.**

To address the above challenges, we present Aeromancer, a feature-rich workflow manager for running MapReduce-based workflows that utilizes both client and cloud resources. Aeromancer offers an ensemble of features, including the simultaneous use of client resources (e.g., on-premises clusters) and public cloud resources; automatic data-dependency and data-transfer handling; intra-flow, on-demand cluster provisioning; and support for directed-acyclic graphs (DAGs). To demonstrate its functionality, we apply Aeromancer to several bioinformatics pipelines, as part of a "big data" case study in the life sciences, which seeks to increase the adoption of hybrid computing environments, including the emerging "client+cloud" computing model, for running data-intensive workflows.

I. Introduction

Cloud computing offers a model where users have on-demand access to computing capability without the institutional overhead of establishing, operating, and updating one's own computing infrastructure. However, domain scientists are often reluctant to embrace cloud computing as a "big data" computing environment for a myriad of concerns, including data security, data-transfer overhead, and ease of use, particularly with respect to configuring and optimizing cloud-enabled applications.

Large-scale data analysis can often be a complex process that involves multiple stages. These stages, in turn, constitute a specific workflow. For example, in bioinformatics, a workflow for genomic variant analysis can encompass the following stages: (1) error correction of raw genomic data, (2) alignment of short reads to a reference genome, (3) local realignment, (4) marking of duplicates, (5) finding polymorphisms, and so on. Managing such workflows, while simultaneously and efficiently using in-house clusters and public clouds, for example, is an arduous and daunting task for many domain scientists. In fact, numerous books (e.g., [1], [2]) have been dedicated just for developing the necessary computer skills in bioinformatics. Our work acknowledges these challenges and presents an easy-to-use workflow manager for running large-scale, data-intensive workloads in the cloud. In this context, a data-intensive workload refers not only to the volume of data that is accessed from storage resources or transferred via the network (I/O rate), but also to the processing of the data (compute rate). Many scientific workloads are both I/O-intensive and compute-intensive.

Hadoop is an open-source MapReduce realization that is used for many such data-intensive workloads. Hadoop has two major subsystems: the MapReduce framework and the Hadoop Distributed File System (HDFS). The MapReduce framework exposes two major primitives: `map` and `reduce`. The map phase takes key-value pairs as inputs and translates them into intermediate key-value pairs. The process continues with reduce tasks that aggregate the results from each map task. HDFS consists of a master Namenode process that manages the filesystem namespace and a Datanode process that runs on each cluster node and stores the file data in its local filesystem. This architecture provides a simple model for parallel data analysis, and hence, is often used to process massive amounts of data in a distributed fashion on large-scale commodity clusters.

While MapReduce provides a blueprint for a computing environment that facilitates ease of execution, we identify additional capabilities that are desirable in the management of MapReduce-based workflows. Specifically, we motivate the need for such capabilities below.

First, not all stages of a workflow are mandatory for every run. Therefore, the manager should provide the flexibility of selecting which stages to run (and not run) via an easy-to-use interface.

Second, not all stages of a workflow exhibit uniform computational needs. Stages with more demanding computational requirements can be spawned in the cloud, where large-scale processing power is available, while stages with less demanding computational needs can be run on in-house clusters, thus saving money on cloud expenses. A workflow manager should provide capability for adaptive choice of client/cloud resources depending on computational needs.

This work was supported in part by NSF CCF-1048253, as part of the NSF Computing in the Cloud Program with Microsoft, and NSF IIS-1247693. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or Microsoft. We thank NSF for funding this project and Microsoft for providing us the hardware to run our experiments and for their timely help in resolving technical issues faced by us.

Third, not all stages of a workflow scale equally. For stages that exhibit relatively poor scalability, performance might not improve much despite an increase in compute resources. As a consequence, cloud costs could be reduced or eliminated by running on fewer nodes in the cloud or running on in-house cluster resources, respectively. This, in turn, points to the need for on-demand provisioning of compute resources in a workflow.

Fourth, some stages might process sensitive data, and hence, should be executed on private clusters for security purposes. A workflow manager should support such adaptability on the fly.

Fifth, special-purpose hardware, such as GPUs and FPGAs, could accelerate computation in the cloud, but such hardware may not be available in some cloud environments. A workflow manager should be able to allocate such functionality on the fly, when such resources are available.

Sixth, if some stages of a workflow can run in the cloud while other stages on an in-house cluster, the workflow manager needs to enable the concurrent execution of such stages, while abiding by any data dependencies between stages. Furthermore, the workflow manager should facilitate any necessary data transfers between the cloud and cluster automatically.

Towards addressing the above issues, we present Aeromancer, a feature-rich workflow manager to configure and monitor multi-stage MapReduce-based pipelines. Aeromancer is built atop of an existing software Cloudgene [3], which provides a graphical user-friendly interface to run a scientific workflow either entirely on an in-house cluster or entirely on public cloud. Additionally, Aeromancer offers a set of features that address the above-mentioned scenarios. The feature set includes the following:

1) Directed-acyclic graph (DAG) support for inter-stage dependencies
2) Support for hybrid environments, i.e., client+cloud resources
3) Automatic data transfers
4) Intra-flow on-demand cluster provisioning in the cloud
5) Facilitating easy debugging;
6) Variability of workflow execution
7) Plugin support;
8) Data transfer optimizations

The rest of the paper is organized as follows. In section II, we present an overview of related work and articulate how our work differs. Section III describes the software architecture of Aeromancer, whose main features are presented in detail in section IV. Finally, in section V, we demonstrate the applicability of Aeromancer in real-world scientific applications for genomic data analysis.

II. Related work

Recently considerable amount of work has been done and published in cloud computing and Hadoop-based workflow management. We start our discussion with Cloudgene [3] on which Aeromancer is built upon. Cloudgene was designed to improve the usability of MapReduce programs in bioinformatics. It is easy to configure and intuitive to use and provides a graphical interface which is a much appreciated feature among domain scientists. Cloudgene is designed using components that are platform-independent like ExtJS[4], Restlet [5], and JSON [6] and offers a rich set of features. Aeromancer extends this feature set by introducing DAG-based support for workflows with intra-stage data dependencies, facilitating the usage of compute resources both in-house and in the cloud simultaneously with automated data transfer, enabling per-stage on-demand cluster provisioning, and many more features mentioned later in the paper. Cloudgene will be discussed in detail in the next section.

In addition to Cloudgene, there exist many software management systems for Hadoop-based pipelines. Kepler [7], one of the earliest efforts to integrate with Hadoop, provides an easy-to-use interface to compose, execute, and monitor MapReduce applications in workflows. On the other hand, Kepler requires low-level configuration that is not very intuitive to use for domain scientists. In addition, Kepler does not integrate with existing cloud providers and does not support on-demand provisioning.

Commercial cloud providers, like Amazon and Microsoft, provide their own workflow managers, Amazon EMR [8] and Microsoft Azure HDInsight [9] with Oozie support [10], respectively. Amazon EMR only manages its public cloud resource and does not address hybrid environments. It also does not provide support for data dependencies in applications. Oozie is a workflow scheduler that is used to configure, run, and monitor MapReduce jobs. However, it does not implement automated data-dependency handling mechanisms between client and cloud resource in a hybrid environment. In both the cases, the onus falls upon the user to ensure that the data is ready and available for the application to run.

The next set of closely related tools include workflow managers for non-MapReduce-based workflows on cloud. Clovr [11] delivers a portable virtual machine (VM) image that provides several automated pipelines. Clovr VMs can utilize either client or cloud resources, but not both simultaneously. Clovr supports automatic provisioning of cluster resources during pipeline execution and offers customizable VM images that can execute on multiple platforms. Clovr is built using different components like Ergatis [12], a workflow system and Vappio [13], which is built on top of the Amazon EC2 API [14] and is used for managing EC2 clusters. Clovr currently runs on grid resources and is not MapReduce-enabled.

Galaxy [15] and Tavaxy [16] are two popular systems for the design and execution of bioinformatics workflows. Tavaxy combines Taverna [17] and Galaxy sub-workflows and supports simultaneous use of local (client) infrastructure and remote resources using web services and triggers. Galaxy can be used in combination with a cloud resource manager called CloudMan [18] to realize *Galaxy Cloudman* [19]. A recent extension to Cloudman added support for Hadoop and HTCondor [20] frameworks, and thus, is capable of "providing Hadoop as a Service," similar to Microsoft Azure HDInsight. Galaxy Cloudman allows users to launch several instances of Galaxy on a cloud to run the same workflow across several datasets. However, Galaxy does not provide native support for

Hadoop workflows, in which MapReduce-based computations in each stage are distributed across compute nodes.

III. Design and Architecture of Aeromancer

Aeromancer builds atop Cloudgene and extends to its functionality. Here we start by discussing the existing design of Cloudgene, followed by a high-level overview of how its architecture was modified in order to create Aeromancer.

A. Cloudgene Overview

Cloudgene consists of two independent modules, namely Cloudgene-Cluster and Cloudgene-MapRed. Cloudgene-Cluster provides the functionality to instantiate a cluster using public cloud resources, in particular, Amazon EC2. Exercising this functionality requires the user to provide the cluster size, credentials, and other associated information. As the Cloudgene-Cluster module is not part of Aeromancer's design, we do not discuss it any further in this paper.

The client side of Cloudgene is designed as a web application that makes use of the JavaScript framework called Sencha Ext JS. The server side utilizes the RESTful web framework *Restlet*. The client and server communicate using a platform-independent format for data interchange called JSON, short for JavaScript Object Notation.

Cloudgene-MapRed gets installed automatically on a namenode of a Hadoop cluster, either in-house or provisioned on the cloud via Cloudgene-Cluster. The user configures the MapReduce job or a pipeline of jobs using a YAML file, which we refer to as the manifest file, that should be written once for each application. Cloudgene parses the manifest file and dynamically creates a graphical web interface for the user. The user may enter the relevant input data or application parameters through this web portal to run a workflow. Once the parameter values are provided and the job is submitted, the job parameters and their corresponding values are communicated from the client (browser) to the Cloudgene server.

Next, we present the design of Cloudgene's job submission module because it is a necessary prerequisite to understand the design of Aeromancer. The job inputs submitted from the web client are sent to the Cloudgene-MapRed server. The server creates a job queue, where the jobs submitted by users are enqueued in order, as specified in Fig.1.

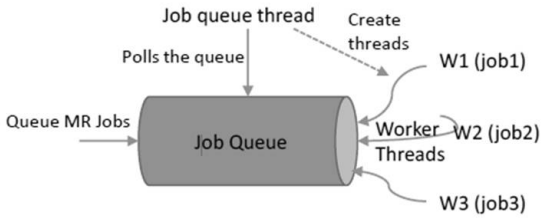

Fig. 1. Cloudgene job submission architecture. Worker thread executes all the stages in the job sequentially.

The enqueued job can be a simple MapReduce job or a MapReduce pipeline consisting of multiple MapReduce jobs or stages. A job-queue thread repeatedly polls the job queue for the arrival of new jobs. As soon as a new job arrives, the job-queue thread spawns a worker thread and associates the worker thread with the newly arrived job. The worker thread dequeues the job and submits it for execution using the available cluster resources. Based on this design, the execution of different jobs are independent and can be processed in parallel by different worker threads. The worker thread writes the job status periodically to a database, which the Cloudgene server queries during the monitoring of the submitted jobs from the client interface. The above design, however, has the following shortcomings:

- In a hybrid environment, where users have in-house clusters in addition to the public cloud resources, Cloudgene cannot utilize both sets of resources simultaneously. It can only lbe configured to use one cluster at a time.
- Cloudgene does not support DAG-based MapReduce pipelines. With the current design, the execution of different stages of the MapReduce pipeline is always sequential.
- The user must ensure that data dependencies are met before executing various stages within a MapReduce pipeline.

B. Overview of the Aeromancer Architecture

Aeromancer is currently designed to work with the Microsoft Azure cloud. Its modular design, however, enables easy integration with other cloud providers, if desired. Future work section includes more detailed discussion of this aspect.

Aeromancer provides *platform as a service (PaaS)* for big data applications running on public clouds. like Azure HDInsight [44], which is a Windows implementation of Apache Hadoop, so user don't need to worry about the entire software stack from the operating system to the application software typical to IaaS. At the same time Aeromancer can be easily deployed on in-house cluster having Hadoop installed, which is rather a rule than exception for company or academic clusters, thus allowing to make use from already existing resources.

Aeromancer has the minimal requirement of having a single-node, on-premises Hadoop configuration. The modified Cloudgene-MapRed server, hereafter referred to as Aeromancer-mapred server, runs on the head node of the on-premises cluster and acts as a driver for the workflow management system. It is responsible for (1) controlling the provisioning of cloud resources, (2) processing workflow DAGs for both control and data dependencies (3) spawning local/remote MapReduce jobs and (4) automatically transferring data between the client and cloud resources to ensure data dependency between different stages of the pipeline. Similar to Cloudgene, the user configures the workflow in a manifest file. The web client module generates graphical wizards to take input from the user. The input is passed to the Aeromancer-mapred server running in the local on-premises cluster. The Aeromancer-mapred server then drives the pipeline execution towards completion. A high-level overview of Aeromancer architecture is shown in Fig.2 The below section discuss the implementation of the features offered by Aeromancer.

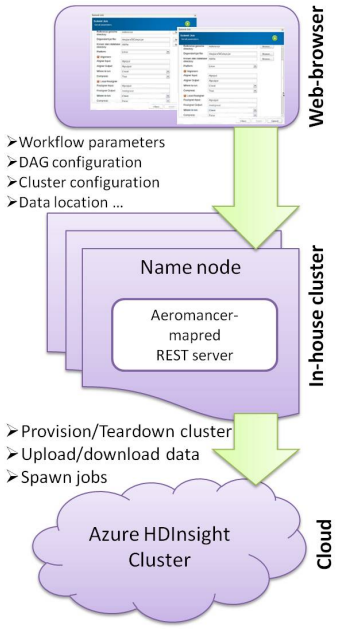

Fig. 2. High level architecture of Aeromancer

```
- name: Table Recalibration
  params: -nfv -nm -nmd -na -nra -nv -ref $reference -b $s1input -o $s1output
          -djarloc $dpjarloc -p  $s3platform -dbfile $dbfile
  alias: s1
  dependency: s0
- name: MergeResults
  params: -nqr -nfv -nmd -na -nra -nv -ref $reference -b $s1output -o $finoutput
          -djarloc $dpjarloc -p  $s3platform
  alias: s2
  dependency: s1
```

DAG Support

Fig. 3. DAG configuration fragment of manifest file

IV. Feature-wise Implementation Details

A. DAG support for inter-stage dependencies

AeroMancer creates a DAG to model the inter-stage dependencies in the workflow. In order to support this feature, the users should be capable of specifying the dependency information between different stages within a MapReduce pipeline. This support is implemented by using the following additional parameters to the manifest file (Fig.3).

1) The *alias* parameter is used to associate each stage in the pipeline with a name. The dependency information can be specified using the alias value of each stage.
2) The *dependency* parameter specifies the control and data dependency information of each stage in the pipeline, if applicable. The value of this parameter specifies a single or a list of alias, which corresponds to other stages in the pipeline that needs to complete before this stage can start its execution.

Once the user submits a job, which corresponds to a MapReduce pipeline, the Aeromancer implementation parses the dependency information for each stage in the pipeline and constructs a directed acyclic dependency graph. The vertex corresponds to a stage in the pipeline and the edges specify the dependency between stages. Only those stages that are enabled by the user at runtime are part of this graph. The dependency graph is constructed by the server thread as soon as the user submits a new job. This job is then enqueued into the job queue. In the traditional Cloudgene architecture, a worker thread dequeues the job and runs the corresponding pipeline in a sequential fashion. In Aeromancer, the worker thread instead queries a DAG processor for the next set of candidate stages to run. Refer to Fig.4. The DAG processor processes the constructed dependency graph and returns the set of vertices or stages that have an in-degree of zero. An in-degree of zero implies that all data and control dependencies are met for that stage. These stages can then run in parallel using both the client and cloud resources simultaneously. Once a particular stage in the pipeline completes, the corresponding vertex is removed from the dependency graph and the worker thread immediately queries the DAG processor for the next set of eligible stages to run.

B. Support for hybrid environments

As discussed earlier, Cloudgene uses a worker thread per MapReduce pipeline and executes all the stages sequentially, using either the client or cloud resources. In order to run multiple stages in parallel using both client and cloud resources simultaneously, Aeromancer introduces the *step queue*, in addition to the job queue in Cloudgene. The step queue sits below the DAG processor and enables asynchronous execution of stages using distributed resources. The set of candidate vertices or stages determined by the DAG processor is fed into this step queue, by the worker thread, for execution. The step queue is periodically polled by a step queue thread. As soon as a vertex or stage is queued, the step queue thread spawns a sub-worker thread for this stage and then continues polling. This results in an asynchronous thread processing each stage of the pipeline in parallel. As a result, multiple stages can be run independently and simultaneously using the desired resources provided by the user. The functionality of how the hybrid cloud support provided by Aeromancer is demonstrated on Fig.4. The sub-worker thread uses Apache Templeton [21] or WebHCat interface to spawn local or remote MapReduce jobs. Templeton provides a REST-based interface for launching both normal and streaming MapReduce jobs. The job launch via WebHCat returns a job ID (Job ID of the Templeton controller job), which can be used to query the status of running jobs.

C. Automatic data transfers

Aeromancer implements automatic data-dependency handling for MapReduce pipelines. In traditional Cloudgene, it is the users responsibility to ensure that the data is available either by manually importing the data or by having the data available from the output of the previous stage. In order to provide this support in Aeromancer, the users should specify the input and output dataset required for each stage in the MapReduce pipeline. This support is implemented using the following two extensions in the manifest file, as shown in Fig.5.

1) The *stepinput* parameter, which is used to specify the input dataset (file or directory) required by each stage in the pipeline. The input data should be prefixed with

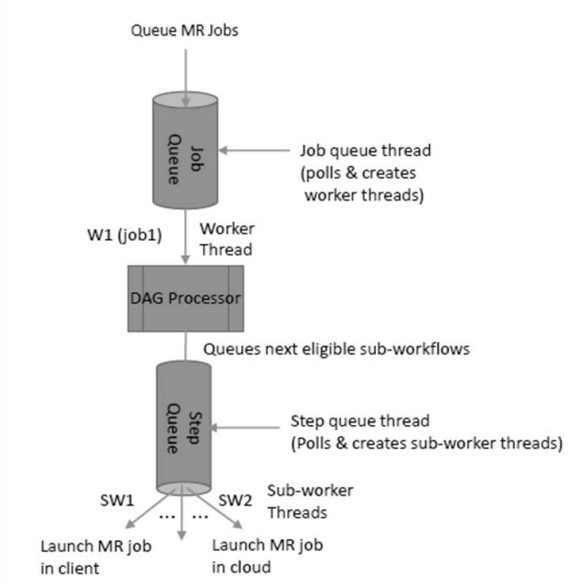

Fig. 4. DAG and Hybrid Support for Aeromancer

```
- name: Table Recalibration
  params: -nfv -nm -nmd -na -nra -nv -ref $reference -b $s1input -o $s1output
          -djarloc $dpjarloc -p  $s3platform -dbfile $dbfile
  stepinput: $reference,$s1input,$dpjarloc,$dbfile
  stepoutput: $s1output

- name: MergeResults
  params: -nqr -nfv -nmd -na -nra -nv -ref $reference -b $s1output -o $finoutput
          -djarloc $dpjarloc -p  $s3platform
  stepinput: $reference,$s1output,$dpjarloc
  stepoutput: $finoutput
```

Data dependency support

Fig. 5. Fragment of manifest file with data-dependency configuration

the appropriate access scheme (hdfs or asv) based on where the data is currently available (HDFS or blob). Blob [22] is a storage service from Microsoft.

2) The *stepoutput* parameter, which is used to specify the output data (file or directory) produced by each stage in the pipeline. The output data gets automatically prefixed with the appropriate access scheme based on where stage is configured to run.

Aeromancer parses the *stepinput* and *stepoutput* values and initializes a global table with this information. This table contains the name of the data and the current location of the data (client, cloud or both). The sub-worker thread consults this table before launching the MapReduce job corresponding to a particular stage in the pipeline. Based on the execution location (either client or cloud) of a stage, the required dataset might be already available in the correct location or Aeromancer automatically transfers it to the correct location.

D. Intra-flow on-demand cluster provisioning on cloud

Aeromancer provides the feature of per-stage on-demand cluster provisioning. The main driver behind Aeromancer is the Aeromancer-mapred server that runs in the client cluster and implements this feature. The number of datanodes to be provisioned can be passed as an input to the Aeromancer interface as shown on Fig.6. For each stage that the user chooses to run on cloud, Aeromancer runs Windows Powershell scripts to check if the cluster has already been provisioned on Azure HDInsight. If not, Aeromancer uses another Powershell script to provision the cluster of size given as an input by the user. Through its interface, Aeromancer also provides the choice of deleting the cluster after the stage has been executed. This feature has currently been implemented only for Microsoft Azure HDInsight and hence works only in Windows environment.

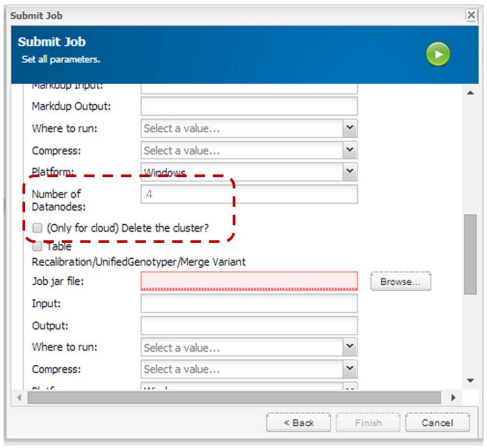

Fig. 6. Snapshot of Aeromancer's interface for on-demand provisioning

E. Facilitating easy debugging

Aeromancer supports running the entire MapReduce pipeline using the local client resources before offloading it to the cloud. This is similar to the support offered by local Oozie runner. This feature ensures that the pipeline runs to completion without issues before provisioning cloud resources. A successful dry run before the actual execution is highly advantageous as it enables one to debug any library, congruration or data availability related issues locally before moving the computation to the cloud. Provisioning cloud resources and then debugging issues incurs unnecessary cost for idle resources.

F. Variability of workflow execution

With the existing design of Cloudgene, the execution of all stages in a MapReduce pipeline is mandatory. Moreover, the entire pipeline executes either on a private cluster or in a public cloud. Aeromancer provides user with the flexibility to enable/disable different stages in a MapReduce pipeline at runtime. In addition, it enables user to specify the execution location of each stage in the pipeline, either the client-cluster or cloud resources. The input data for a stage could either be the output of its previous stage or should be made available by the user in an appropriate location. This feature can be very useful when results of a particular stage are already known and someone wants to reuse them for downstream stages.

G. Plugin support

Certain functionalities for instance data preprocessing before transfer which need not functionally be a part of a particular workflow but an additional feature of the framework,

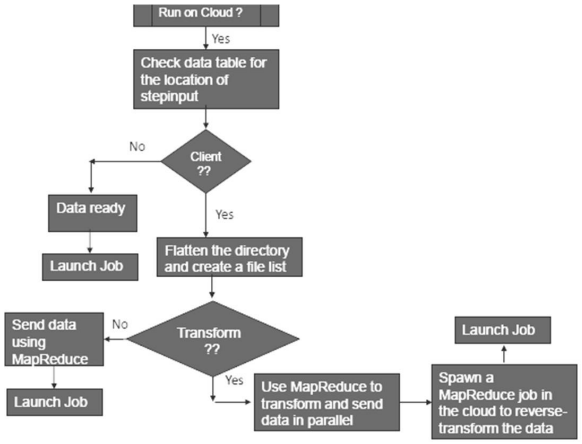

Fig. 7. Aeromancer data transfer flowchart

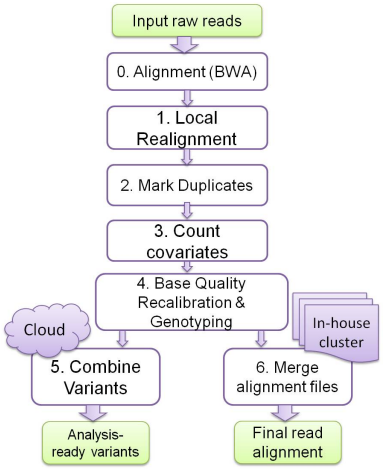

Fig. 8. SeqInCloud workflow

can be added as a plugin support. This can be achieved by having a common stub in the Aeromancer code with some common interfaces exported. These interfaces are implemented in specific libraries. For instance, to support data encryption and compression before transfer, the Aeromancer code will have stubs one for encryption/decryption and one for compression/decompression. User can have their own encryption/decryption or compression/decompression library linked to the framework. The modules from the user-specific library will automatically get invoked and will be used to transform the data before and reverse-transform after transfer happens. Implementation of this feature is currently under development.

H. Data transfer optimizations

Aeromancer optimizes data transfers from client to cloud and vice versa using MapReduce. In addition, MapReduce can be used to optimize the user-defined transformations that are defined by plugins. Using MapReduce for data transfers and transformations results in higher utilization of cluster resources. It also gives us the aggregate bandwidth and aggregate processing capacity of the cluster to transfer and transform data, respectively. The algorithm is presented in the flowchart on Fig.7.

V. Case Study

We have tested the functionality of Aeromancer by integrating several MapReduce-based bioinformatics pipelines, as our group is most familiar with this area, but there are no reasons to limit the use of Aeromancer to bioinformatics only. The tests were carried using Microsoft Azure HDInSight as public cloud which is a PaaS-based big data solution powered by a Windows implementation of Apache Hadoop. HDInsight provides the flexibility of provisioning a cluster with as many datanodes as desired, each datanode being a Virtual Machine with 4 cores. For storing data on cloud, Windows Azure blob services was used [22]. Blob storage is a service for storing large amounts of unstructured data in the cloud accessed via http[s]. Data stored in blob storage containers is accessed using ASV URI (asv://) or WASB URI (wasb://).

For the in-house cluster two different environments were experimented upon: a Linux cluster with 8+1 nodes and a Windows cluster with a single-node Hadoop setup on a Windows 8 machine. The machine had Powershell installed and cmdlets were used to communicate and manage the HDInsight cluster on cloud.

Summarizing actions described in section 3 in order to create a workflow and manage it with Aeromancer one needs to (1) define each stage of the workflow as an independent Hadoop application, (2) create a YAML file with runtime configuration and cloud provider information, then (3) launch the GUI generated by Aeromancer based on the YAML file, (4) using the GUI choose i/o files and define parameters' values for the upcoming run, and (5) start the workflow.

A. Integrating SeqInCloud pipeline

A major share of experiments was done using the SeqInCloud [23] pipeline which is a highly scalable parallel implementation of a popular genetic variants discovery pipeline recommended by Broad Institute [24] and built with various analysis tools, including BWA [25], GATK [26] and Picard [27]. The choice of SeqInCloud for our experiments was biased by the fact that it is a multistage MapReduce pipeline with a good mix of inter-dependent and mutually-independent stages. SeqInCloud seamlessly integrates all the stages with Hadoop. SeqInCloud starts with the *Alignment* stage, which uses distributed implementation of BWA (Fig.8). The aligned reads are sorted, merged, and fed into a *Local Realignment* stage, which uses the RealignerTargetCreator and IndelRealigner walkers from GATK. This is followed by *Mark Duplicates*, *Count Covariates* and *Base Quality Recalibration with Genotyping* stages, which are responsible for the identification and filtering of structural variants. Finally, the found variants are combined in *CombineVariants* stage and distributed fragments with realigned reads are merged in one single file in *Merge Alignment Files* stage. The SeqInCloud pipeline takes FASTQ files with raw reads as an input and emits both structural variants in VCF format and analysis ready reads in BAM format as an output. SeqInCloud was run as a whole pipeline as well as in parts by selecting only certain stages. The *alignment* stage can be run separately using different choice of aligners and the SeqInCloud pipeline can be started by Aeromancer from the *Local Realignment* stage.

```
- name: Recalibration
  alias: s1
  dependency: none
  runlocation: Client
  stepinput: $reference,$s1input,$dbfile,$ipjar
  stepoutput: $s1mergeout, $s1variantout

- name: Merge BAM
  alias: s2
  dependency: s1
  runlocation: Client
  stepinput: $reference,$s1mergeout,$ipjar
  stepoutput: $s2out

- name: Merge Variant
  alias: s3
  dependency: s1
  runlocation: Cloud
  stepinput: $reference,$s1variantout,$ipjar
  stepoutput: $s3out
```

Fig. 9. A snippet of the manifest file for SeqInCloud. Fields: **dependency** - for DAG support, **runlocation** for hybrid resources, **stepinput** and **stepoutput** for automatic data transferring

Name	Location
$reference	Client
$s1input	Client
$dbfile	Client
$ipjar	Client
$s1mergeout	Client
$s1variantout	Client
$s2out	Client
$s3out	Cloud

a

Name	Location
$reference	Both
$s1input	Client
$dbfile	Client
$ipjar	Both
$s1mergeout	Client
$s1variantout	Both
$s2out	Client
$s3out	Cloud

b

Fig. 10. a) Data-dependency table with the initial values; b) Data-dependency table after data transferring

Described below is the exhibition of Aeromancer's features using SeqinCloud.

1) On-demand provisioning: Usually a cost of the cloud services is proportional to the number of CPU-hours, measured as wall time multiplied by the number of provisioned compute instances. Based on profiling the execution time of SeqInCloud stage by stage, we observed that the scalability of different stages varied significantly. For example, for input BAM file of size 840MB we found that the reasonable number of compute nodes to provision is: 1 node for stages 1, 5 and 6, 2 nodes for stage 2, and 5 nodes for stages 3 and 4. The resulting wall time was 73 minutes plus time for cluster re-provisioning, while it took 153 CPU-minutes. In comparison with running the stages using all the 5 nodes and consuming a 69 minutes of wall-time and 349 CPU-minutes, using only the reasonable number of nodes halved the paid CPU-minutes with a negligible increase in wall time.

2) DAG and Hybrid Cloud Support: The stages in SeqInCloud are sequential in nature except the last stage where the merging of the BAM output (".bam" fragments) and merging of variant output (".vcf" files) can be run simultaneously. The merging of the BAM fragments is done by a single process that runs on the head node of the cluster. Moving this computation to the cloud does not help as the scale of the cluster is immaterial here. However, running the *Combine Variants* stage using large-scale resources like the cloud will be beneficial. This is a good example of hybrid resource usage as the merging of BAM fragments and merging of variant files will run simultaneously using both client and cloud resources. A snippet of the YAML file for this configuration is shown on Fig.9.

3) Automatic Data-Dependency Handling: Let us assume that the *Base Quality Recalibration and Genotyping* stage is configured to run using on-premise resources. The *Merge alignment files* step is configured to run using on-premise resources, and the *Combine variants* step is configured to run on cloud resources. As, shown in a snippet of the manifest file (Fig.9) input and output parameters for these stages, *stepinput* and *stepoutput* respectively, take values, which marked with a $ sign from a user via the generated user interface. As described earlier Aeromancer parses the manifest file and initializes a global table. Initially, the global table looks as shown in Fig.10a. After stage s1 completes, the DAG processor returns s2, s3 as the candidate stages to run. As configured by user, s2 will be run on client resources and s3 on cloud resources. The data-dependency module parses *stepinput* and *stepoutput* parameters and triggers the data transfer as required. In our case, the stage s2 has all the required data available at the client. But for stage s3, the input needs to be transferred to the cloud before the stage could run. The data-dependency module makes use of the MapReduce framework to transfer data in parallel to the cloud. After the completion of data transferring, the global table is changed as shown in Fig.10b.

B. Other workflows

Aeromancer has also been experimented with two other popular bioinformatics workflows for genome assemly, namely Contrail [29] and CloudBrush [28].

Contrail uses Hadoop for *de novo* assembly of large genomes from short sequencing reads. *de novo* assemblers are usually memory and compute intensive. Contrail relies on Hadoop to iteratively transform an on-disk representation of the assembly graph, allowing an in depth analysis even for large genomes. Contrail workflow consists of several stages, but currently only three of them are available [30]: (1) Converting FastQ files into AVRO format files, (2) building the de Bruijn graph, (3) QuickMerge. Contrail has been successfully integrated in Aeromancer and tested on various sizes of input datasets. The second stage of Contrail ran into I/O failures when using multiple nodes, even when launching directly without Aeromancer, and this restricted us to provision only a single node. Aeromancer's on-demand provisioning feature was used to provision multiple nodes for first and third stages and a single node for second stage to complete the execution without failures. Some pre-processing stages, like extension of short reads and error correction may also be added to the *de novo* assembly pipeline with Aeromancer upon the availability of corresponding MapReduce implementations.

CloudBrush is a newer distributed genome assembler based on string graphs and MapReduce framework. CloudBrush workflow consists of two main phases, namely Graph Construction and Graph Simplification. Graph Construction includes four stages: (1) retaining non-redundant reads as vertices, (2) finding pairwise overlaps between reads, (3) edge adjustment and (4) reducing transitive edges. For Graph Simplification, (5) path compression, (6) tip and bubble removal, and (7) low coverage node removal stages are mandatory. In [28] it is noticed that Graph Construction phase takes significantly more time than Graph Simplification, but on the other hand with an increase in the number of nodes, the computation time of Graph Construction decreases substantially, while the Graph Simplification phase scales only slightly. On-demand provisioning feature of Aeromancer allows one to choose only

reasonably required number of cloud nodes for each stage. Graph Simplification phase also can be optionally offloaded to a local cluster.

VI. Conclusion and future work

Our future plans with Aeromancer include (1) support of different cloud-providers: Since Aeromancer is a ExtJS and REST-based server implementation, the majority of its features are platform-independent, such as DAG support, hybrid cloud support and variability of workflow. As such and to facilitate broader adoption, Aeromancer is architected in a modular fashion so that it is easier to extend to a variety of other cloud provider environments. Currently, the only aspects of Aeromancer that are specific to Microsoft HDInsight are the data-management interface and the on-demand cloud provisioning interface. The former uses the Azure Blob API to make the data available in the Azure Blob for running MapReduce jobs. The latter uses the Azure management API to provision cloud resources. Thus, in order to extend Aeromancer to other cloud environments, the data-management interface and on-demand cloud provisioning interface specific to cloud-providers need to be instantiated. (2) Extension of Aeromancer to support a mix of MapReduce and non-MapReduce stages in a workflow (3) Enhance Aeromancer with the ability to predict the compute resources needed for each stage of the workflow based on its input data size and provide intelligent suggestions for the user about the optimal choice of location and size of cluster to run the workload. This suggestion should take into consideration the predicted computation needs, network costs, storage costs and client and cluster configurations.

Aeromancer is easy to use and hides from domain scientists the complexity of MapReduce job management, resource allocation and data transferring. The combination of its different features makes Aeromancer a superior tool for management of MapReduce based workflows and we believe that our work will increase the adoption of cluster and cloud resources for data-intensive applications in scientific domains.

Acknowledgment

The authors would like to thank Dr. Heshan Lin for his guidance and Microsoft for their support in the form of infrastructure and counsel.

References

[1] C. Gibas and P. Jambeck, *Developing Bioinformatics Computer Skills*, 1st ed. O'Reilly Media, Apr. 2001. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1565926641

[2] S. Haddock and C. Dunn, *Practical Computing for Biologists*, 1st ed. Sinauer Associates, Inc., Nov. 2010. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0878933913

[3] S. Schonherr, L. Forer, H. WeiSZensteiner, F. Kronenberg, G. Specht, and A. K. Brandstatter, "Cloudgene: A graphical execution platform for MapReduce programs on private and public clouds," *BMC Bioinformatics*, vol. 13, no. 1, pp. 200+, Aug. 2012. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-13-200

[4] ExtJS. [Online]. Available: http://www.sencha.com/products/extjs

[5] Restlet. [Online]. Available: http://restlet.org

[6] Json. [Online]. Available: http://www.json.org

[7] 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: Research Articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, Aug. 2006. [Online]. Available: http://dx.doi.org/10.1002/cpe.v18:10

[8] Amazon EMR. [Online]. Available: http://aws.amazon.com

[9] *Microsoft Azure HDInsight*. [Online]. Available: http://azure.microsoft.com/en-us/documentation/services/hdinsight

[10] Oozie. [Online]. Available: http://oozie.apache.org/

[11] S. Angiuoli, M. Matalka, A. Gussman, K. Galens, M. Vangala, D. Riley, C. Arze, J. White, O. White, and W. F. Fricke, "CloVR: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing," *BMC Bioinformatics*, vol. 12, no. 1, pp. 356+, Aug. 2011. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-12-356

[12] C. Hemmerich, A. Buechlein, R. Podicheti, K. V. Revanna, and Q. Dong, "An Ergatis-based prokaryotic genome annotation web server," *Bioinformatics*, vol. 26, no. 8, pp. 1122–1124, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btq090

[13] Vappio. [Online]. Available: http://vappio.sf.net

[14] Amazon EMR API. [Online]. Available: http://docs.aws.amazon.com/ElasticMapReduce/latest/API/Welcome.html

[15] B. Giardine, C. Riemer, R. C. Hardison, R. Burhans, L. Elnitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor, W. Miller, W. J. Kent, and A. Nekrutenko, "Galaxy: a platform for interactive large-scale genome analysis." *Genome research*, vol. 15, no. 10, pp. 1451–1455, Oct. 2005. [Online]. Available: http://dx.doi.org/10.1101/gr.4086505

[16] M. Abouelhoda, S. Issa, and M. Ghanem, "Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support," *BMC Bioinformatics*, vol. 13, no. 1, pp. 77+, May 2012. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-13-77

[17] 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, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/bth361

[18] E. Afgan, B. Chapman, and J. Taylor, "CloudMan as a platform for tool, data, and analysis distribution," *BMC Bioinformatics*, vol. 13, no. 1, pp. 315+, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-13-315

[19] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor, "Galaxy CloudMan: delivering cloud compute clusters," *BMC Bioinformatics*, vol. 11, no. Suppl 12, pp. S4+, 2010. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-11-s12-s4

[20] HTCondor. [Online]. Available: http://research.cs.wisc.edu/htcondor

[21] Apache Templeton. [Online]. Available: http://people.apache.org/~thejas/templeton_doc_latest/

[22] Windows Azure Blob. [Online]. Available: http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-blobs/

[23] N. M. Mohamed, H. Lin, and W.-c. Feng, "Accelerating Data-Intensive Genome Analysis in the Cloud," in *5th International Conference on Bioinformatics and Computational Biology (BICoB)*, Honolulu, Hawaii, USA, March 2013.

[24] Broad Institute. [Online]. Available: http://www.broadinstitute.org

[25] BWA. [Online]. Available: http://bio-bwa.sourceforge.net

[26] GATK. [Online]. Available: https://www.broadinstitute.org/gatk

[27] Picard. [Online]. Available: http://picard.sourceforge.net

[28] Y.-J. J. Chang, C.-C. C. Chen, C.-L. L. Chen, and J.-M. M. Ho, "A de novo next generation genomic sequence assembler based on string graph and MapReduce cloud computing framework." *BMC genomics*, vol. 13 Suppl 7, no. Suppl 7, pp. S28+, 2012. [Online]. Available: http://dx.doi.org/10.1186/1471-2164-13-s7-s28

[29] Contrail. [Online]. Available: http://contrail-bio.sf.net

[30] Contrail Quickstart. [Online]. Available: http://sourceforge.net/apps/mediawiki/contrail-bio/index.php?title=Quickstart