Integrating Data-Intensive Computing Systems with Biological Data Analysis Frameworks

Edvard Pedersen[1], Inge Alexander Raknes[2], Martin Ernstsen[1*], Lars Ailo Bongo[1]

[1]Department of Computer Science and
Center for Bioinformatics
University of Tromsø, Norway
[*]Now at Kongsberg Satellite Services AS, Tromsø, Norway
edvard.pedersen@uit.no, martin.ernstsen@ksat.no,
larsab@cs.uit.no

[2]Norstruct,
Department of Chemistry,
University of Tromsø, Norway
inge.a.raknes@uit.no

***Abstract*—Biological data analysis is typically implemented using a pipeline that combines many data analysis tools and meta-databases. These pipelines must scale to very large datasets, and therefore often require parallel and distributed computing. There are many infrastructure systems for data-intensive computing. However, most biological data analysis pipelines do not leverage these systems. An important challenge is therefore to integrate biological data analysis frameworks with data-intensive computing infrastructure systems. In this paper, we describe how we have extended data-intensive computing systems to support unmodified biological data analysis tools. We also describe four approaches for integrating the extended systems with biological data analysis frameworks, and discuss challenges for such integration on production platforms. Our results demonstrate how biological data analysis pipelines can benefit from infrastructure systems for data-intensive computing.**

Keywords—biological data processing, data-intensive computing, integration

I. Introduction

Recent advances in instrument, computation, and storage technologies have resulted in large amounts of biological data [1]. To realize the full potential for novel scientific insight in the data, it is necessary to transform the data to knowledge through data analysis and interpretation. Such biological data analysis is typically implemented using a data analysis pipeline that combines a set of tools and databases. To answer a specific biological question the data analysis often requires a unique combination of tools. There are many libraries (such as BioConductor [2]), and hundreds of available tools (as in Galaxy Toolshed [3]). These range from small, user-created scripts to large, complex applications.

To use such pipelines, the analyst specifies, configures, and executes the pipeline using a biological data analysis framework [2], [4], [5]. These provide a way of specifying the pipeline tools and their arguments, management of pipeline data and meta-data, and execution of the pipeline tools. The data analysis framework is often implemented on top of infrastructure systems for data management and parallel job execution.

In our experience, most biological data analysis pipelines are run on a high-performance computing platform that consist of many compute nodes, connected with a fast interconnect, and a centralized storage solution. The platform provides a network file system for storage, databases for data management, a parallel programming model such as MPI [6] and OpenMP [7], parallel data management libraries such as PnetCDF [8] and PHDF5 [9], and a parallel job execution system such as Open Grid Engine [10] or Torque [11]. This platform has four main advantages. First, the file system-, and database interfaces are stable, and the technology is reliable. Second, many clusters for scientific computation are designed to use a network file system and to run jobs on a cluster using a queuing system. Third, the developers and administrators are familiar with these systems and interfaces. Fourth, there are already hundreds of tools implemented for this platform.

However, we believe this standard platform will not scale to the upcoming biological datasets as demonstrated for other large-scale data analysis projects [12], [13]. It will therefore become necessary to implement biological data analysis frameworks on infrastructure systems and hardware platforms designed for data-intensive computing [14], [12], [13], [15], [16]. Currently most biological data analysis frameworks do not leverage such systems and platforms.

We believe that in order for biological data analysis frameworks to leverage data-intensive computing systems they must overcome four challenges. First, the hardware platform must use distributed data storage (as in [17], [12]). Second, the systems must be stable and reliable. Third, the systems must run the many existing analysis tools with no or minor modifications. Finally, the data-intensive infrastructure systems must be integrated with biological data processing frameworks.

To address these challenges we have extended data-intensive computing systems with services for biological data processing. Our approach does not require any changes to analysis tools. We have integrated these services with three biological data processing frameworks using four approaches. We have implemented services that only require a small computer clusters. Such clusters are realistic to purchase by individual labs. We build on the widely used Apache Hadoop platform [14] that is both stable and reliable. Combined our work demonstrates how unmodified biological data analysis tools can benefit from infrastructure systems for data-intensive computing.

II. Related Work

We discuss the limitations and lessons learned utilizing data-intensive systems for biological data processing in [18].

A. Data-Intensive Computing Systems

There are several specialized infrastructure systems developed for data-intensive computing. Many of these were initially developed and deployed at web-scale companies such as Google, Yahoo, Facebook, and Twitter, and then later implemented as open source systems. There are also many new systems under development in academia, the open source community, and the industry.

Data intensive computing systems are often built on a distributed file systems such as Hadoop Distributed File System (HDFS) [17] that provide reliable storage on commodity component hardware and high aggregate I/O performance. A HDFS cluster co-locates storage and computation resources to avoid the bandwidth limitation of storage area networks when transferring large datasets over the network to and from the computation nodes. For large systems, it reduces cost since high-volume network attached storage is expensive. The main advantage of HDFS for biological data analysis is that the architecture is demonstrated to scale to peta-scale datasets, and it is widely used for data-intensive computing in other fields. The main disadvantage is that HDFS does not provide a traditional file system interface, so it is necessary to either rewrite the many data analysis tools that use a POSIX file system interface, to incur an overhead for moving data between HDFS and a local filesystem, or incur the overhead of third-party library such as fuse-dfs [19]. In addition, it is not yet a common platform in scientific computing, so it may be necessary to purchase and build a new cluster with storage distributed on the compute nodes instead of a centralized storage system.

MapReduce [13][14] is a widely used programming model and infrastructure system for data-intensive parallel computation. It provides fault-tolerant computation on a HDFS-like file system, and makes it easy to write scalable applications since the system handles data partitioning, fault-tolerance, scheduling, and communication. Biological data analysis applications, especially for next-generation sequencing data, have already been implemented using MapReduce ([20] provides examples). The main advantage of MapReduce is that it scales to peta-scale datasets. In addition, most cloud platforms provide a MapReduce interface. The main disadvantage is that the MapReduce programming model may not be suited for all biological data analysis applications.

An alternative for low-latency query processing is Spark [16][21]. It offers a richer programming model than MapReduce, including iterative operations. It is well suited to implement machine learning algorithms, and interactive data analytics. However, Spark uses the Scala programming language, which may be unfamiliar to many developers In addition, compared to the systems discussed above Spark has just recently become a top-level Apache project, but it is rapidly being adopted by many other open-source and commercial systems.

HBase [22], [15] is a column based storage system that provides in-memory caching for low latency random data access and efficient compression. Biological data analysis applications can use HBase to store data accessed interactively, to implement custom data structures, or to structure data for more efficient compression. Compared to relational databases, HBase does not provide an advanced query engine nor ACID properties, so these must be implement on top of HBase if needed.

Several high-level programming models are built on top of MapReduce to make it easier to write data analysis programs, including Pig [23], Hive [24], Cascading [25], Cassandra [26], and the Mahout [27] library of machine learning algorithms. But, to our knowledge these are not widely used for biological data analysis.

B. Biological Data Processing Frameworks

There are many frameworks for specifying and running biological data analysis pipelines. The two most widely used systems are Galaxy [4] and Taverna [5].

Galaxy is a tool for biomedical research, and consists of a framework for running tools, a web server, and a web-baed interface for adding tools to the framework. The web interface allows the user to runs tools or pipelines on the cluster or computer connected to the Galaxy server. The administrator of the Galaxy server can add new tools to the Galaxy server by creating small XML scripts that describe how to run the tool, which parameters, input data and output data are required by the tool. CloudMan [28] is an extension to Galaxy, which enables the use of cloud infrastructure, such as Amazon EC2, for executing Galaxy workflow jobs.

The Taverna Workbench is a general framework for creating, managing and running workflows of tools. The user can interact with Taverna through a web interface, a desktop tool or a command-line tool. The user specifies a workflow using the custom Scufl language. It is possible to export Taverna workflows as Galaxy tools.

In addition there are many general purpose high-performance systems that can be used to run biological data processing pipelines, such as cluster schedulers [10], Grid schedulers [29], cluster managers [30], and systems for distributed application deployment [31]. However, these are typically not integrated with neither Galaxy nor Taverna (although it can be done as described in the next section and in [32]).

C. Cloud Computing Platforms

Cloud computing platforms are an emerging platform for biological data processing. Commercial clouds store large-scale biological datasets (as in Amazon AWS [33]), and provide the compute resources for analyzing the datasets (e.g. Amazon EC2). There are also cloud solutions such as the Embassy Cloud [34] for processing the large data repositories at the European Bioinformatics Institute (EBI).

Cloud services such as EC2 and Embassy Cloud run virtual machines provided by the user in very large data-centers. The user only pays for the resources needed. There are three main advantages of cloud computing for large-scale biological data. First, the user gets access to a very large compute cluster designed for data-intensive computing. Second, the cloud provides the resources and elasticity needed to scale the job for very large datasets. Third, the processing can be done close to data. A disadvantage is the cost of resources, which is often higher than in supercomputing centers. In addition, cloud platforms are generally not as optimized for parallel programs as high-performance computing platforms.

Our systems and integration approaches are well suited for cloud computing.

III. Data-Intensive Computing Services for Biological Data Analysis

We have extended several data-intensive computing systems to provide services for biological data analysis. In this section, we describe how our biological data processing pipelines benefit from data-intensive processing services. In the next sections, we describe and discuss integration with biological data analysis frameworks.

We have used clusters built for data-intensive computing with storage distributed on the compute nodes. We use the Hadoop software stack. It is probably the mostly used platform for data-intensive computing, and it has a very active development community.

A. Troilkatt

Troilkatt is a system for batch processing of large-scale collections of gene expression datasets. We use Troilkatt to process data for the IMP integrated data analysis tool [35], [36]. We built Troilkatt in order to scale our gene expression dataset integration pipeline to process all datasets for several organisms in NCBI GEO [37]. The pipelines comprise tools for data cleaning, transformation, and signal balancing of these datasets. The integrated data compendium for organisms such as human have ten thousands of datasets. But, these can be processed independently. The raw data, results, and intermediate data in a compendium use tens of terabyte of storage space.

Troilkatt provides infrastructure services for automated genomics compendium management. It consists of five main components (figure 1). First, the large genomics compendia maintained by Troilkatt are stored and processed on a cluster. Second, Troilkatt leverages the Hadoop software stack for reliable storage and scalable fault-tolerant data processing, including the Hadoop Distributed File System (HDFS), Hadoop MapReduce, and HBase. Third, the Troilkatt runtime system provides data management including versioning and a library of tools for downloading and processing data. Fourth, Troilkatt can execute a large collection of external tools and scripts for various data processing. Finally, a command line based user interface allows the administrator to control the compendium content and data processing.

An expert user specifies a Troilkatt pipeline as an XML file. The file specifies the tools to run, their arguments, datasets, and meta-databases. In addition, the user tunes pipeline performance and data storage requirements by specifying storage systems, parallel data execution framework, compression algorithms, and data retention time. The user does not implement communication between tasks, data locality aware mapping of tasks to compute nodes, or fault-tolerance. The Hadoop software stack handles these.

TABLE I. Troilkatt summary.

Goal	Scalable integrated data analysis pipelines
Systems used	HDFS: scalable storage MapReduce: I/O intensive pipeline processing
Tools	Unmodified binaries and scripts MapReduce applications

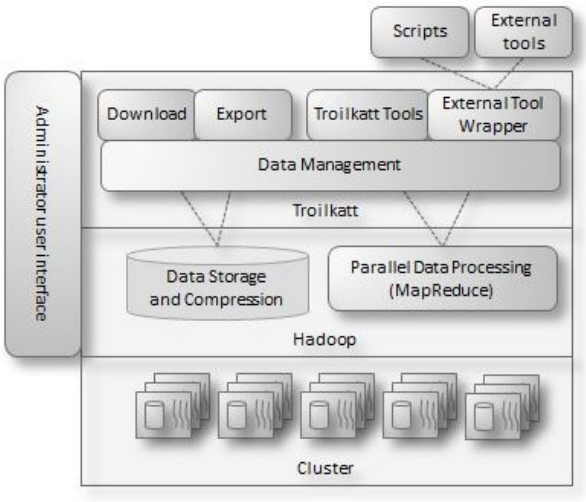

Fig. 1. Troilkatt architecture.

Troilkatt supports three types of tools: unmodified binaries or scripts, Troilkatt scripts, and MapReduce jobs. Troilkatt executes unmodified binaries in parallel using MapReduce. This is similar to Hadoop Streaming, but Troilkatt allows specifying additional command line arguments as environment variables. For example, many biological analysis tools require specifying multiple meta-database files as command line arguments. The disadvantage of using unmodified binaries is that Troilkatt must copy input and meta-database files to a local file system before executing the tool, and then after tool execution, copy the result file back to HDFS.

Troilkatt scripts provide classes and interfaces that make it easy to implement for example data retriever scripts. Troilkatt MapReduce jobs provide optimized access to the data stored by Troilkatt. It is well suited for many of the IMP data processing tools, since they process one row in a dataset at a time. Both use the HDFS interface to directly read and write data to files in HDFS.

B. GeStore

GeStore [38] is a system for adding transparent incremental updates to biological data processing pipelines. We use GeStore to periodically update large-scale compendia, such as the IMP compendia described in the previous section. We built GeStore since the processing time for a full compendium update can be several days even on a large computer cluster, making it impractical to frequently update large-scale compendia. We have achieved up to 82% reduction in analysis time for dataset updates when using GeStore with an unmodified biological data analysis pipeline [38]. GeStore also provides efficient meta-database management for large scale meta-databases.

TABLE II. GeStore summary.

Goals	Incremental updates for analysis pipelines Meta-data management
Systems	HBase: data structures for generating incremental updates MapReduce: I/O intensive pipeline processing
Tools	Unmodified binaries

GeStore provides a transparent file based approach that allows using unmodified biological analysis tools. Instead, we modify the input and meta-data files read by analysis tools such that these only contain the data for incremental update computations. Executing the tool will typically produce a partial result, which GeStore merges with previously produced results. We have chosen a file based approach since relatively few file formats are used by many genomics applications. It is therefore feasible to implement parsers that support most file formats and therefore most genomics pipeline tools. In addition, most file formats are simple and structured which makes it easy to implement parsers.

GeStore provides services to pipeline managers, and is therefore usually transparent for the end user. The pipeline manager uses GeStore as a storage backend, instead of-, or in addition to the network file system.

GeStore must detect changes in, and merge updates into, compendia that can be tens of terabytes in size. GeStore must also maintain and generate incremental updates or versions of meta-databases such as UniProt [39]. These can be hundreds of gigabytes in size. We use HBase for data storage and MapReduce for generating input files and merging output files. HBase provides high-throughput random data accesses required for efficient change detection and merging. GeStore stores meta-database entries as HBase rows, and updates to these entries by creating a new version of an HBase table cell. The timestamps enable efficient table scans to find entries that have changed in a time period and hence are part of an incremental update. In addition, the flexible schema of HBase tables is used to reduce the work required to maintain plugins when file structures or databases changes, allowing several years of database versions to be stored in the same HBase table. In addition, we reduce storage space requirements by storing multiple meta-database versions in HBase instead of storing all versions separately.

GeStore provides a plugin system to support many biological tools and file formats (figure 2). To add incremental updates for a new tool the plugin maintainer implements a plugin. The plugin specifies how to partition input and meta-data files into entries, which part of an entry are required for the analysis done by a tool, and how to compare the entries to detect updates. In addition, the plugin contains code for writing entries to an incremental input file, and code to merge incremental output with previously produced results. These plugins are relatively small in size; less than 300 lines of code in our most complex plugin. Plugins can access data directly in HDFS or Hbase to reduce overhead.

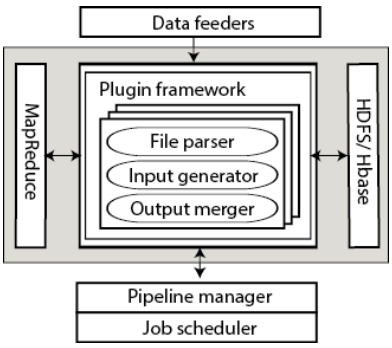

Fig. 2. GeStore architecture.

TABLE III. MARIO SUMMARY.

Goals	Interactive tuning of analysis pipeline
Systems	HBase: sparse data structure, low- pipelines latency reads and writes
Tools	Unmodified binaries Modified data-exploration tools

C. Mario

Mario [40] is a system for interactive iterative data processing (figure 3). We have designed Mario for interactive parameter tuning of biological data analysis pipeline tools. For such workloads, the pipeline output should be visible for the pipeline developer as soon as it is produced. Mario combines reservoir sampling, fine-grained caching of derived datasets, and a data-parallel processing model for quickly computing the results of changes to pipeline parameters. It adds less than 100ms of overhead per pipeline stage, and it does not add significant computation, memory, or storage overhead to compute nodes [40].

Mario must efficiently produce random samples from a stream for reservoir sampling, implement a cache of fine-grained pipeline tool results, and implement parallel pipeline stage processing. We use HBase as storage backend due to its low-latency random read and write capability, its ability to efficiently store sparse data structures, and its scalability. Mario stores all intermediate data records produced during pipeline execution in HBase, and uses the cached data to quickly find the data records that must be updated when pipeline tool parameters are changed. Mario also uses HBase for data provenance and single-pass reservoir sampling. The iterative processing in Mario is similar to Spark Streaming [41]. Mario splits the data to be processed randomly into many small parts, and distributes these on the cluster nodes. The parts are processed in parallel, but since there are many more parts than processor cores, only a small subset is processed at a time. The output data is therefore incrementally updated.

To analyze a dataset using Mario, a user would first load the input data into HBase. She would then use a data analysis framework such as Galaxy [4] to define the pipeline by specifying, for each pipeline stage: the tool to execute, the tool version, and the tool parameters. The configuration can specify that a dataset should be sampled with a given sample size. Mario uses the GeStore transparent file-based approach for incremental processing [38], so it is not necessary to modify

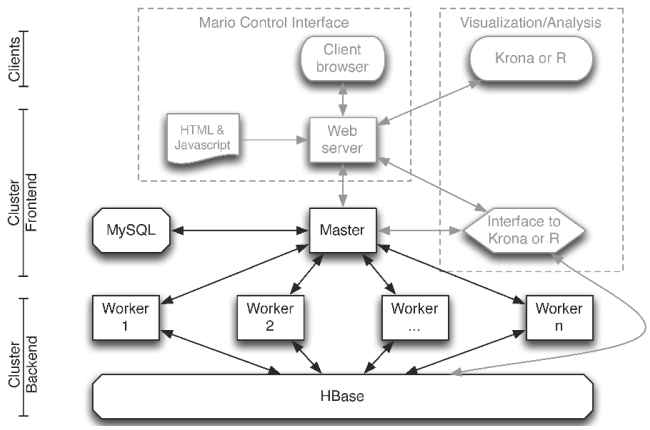

Fig. 3. Mario architecture.

tool code. The user then starts the initial computation. As the computation proceeds, the user can change the parameters or tool used in a stage by sending an updated configuration to the master, which will start scheduling work with the new configuration. If the new configuration does not produce satisfactory results, Mario can restore a previous configuration by reading data for a previous configuration from the storage layer.

A data exploration tool uses an interface provided by Mario to retrieve results from HBase, either periodically or when notified of the presence of new results by the Mario master. To integrate Mario with a visualization system such as Krona [42] or METAREP [43], an interface must be implemented that updates the data structures used by the exploration tool. For Krona, this involves generating an XML-file of the organism hierarchies found in the data. For METAREP, a search engine data structure must be updated. For integration with end-user statistical analysis, the interface can be implemented in for example R.

IV. Biological Data Processing Framework Integration

We describe how we integrated the GeStore system with three pipeline managers (table V) using four approaches (table VI). All approaches add incremental updates to biological data analysis pipelines using the GeStore transparent file-based approach.

Each approach requires adding a call to GeStore before each tool in the pipeline is executed to generate input and meta-data files for the tool. Then, when the tool has executed, GeStore must be called to merge generated output files with previously generated files. For each approach we want to answer the following questions:

- Is GeStore functionally visible to the pipeline user?
- What is the required developer effort for the integration?
- Can the framework use all features of GeStore?
- What is the performance overhead of the approach?

Ideally, GeStore should be hidden from the user since we assume that the biologists using the pipeline do not want to be exposed to pipeline execution and data management details. The developer effort should be small such that we can easily integrate GeStore with the biological data processing framework. We also want to reduce the amount of work required for redoing the integration each time the framework code is updated. In addition, the integration should expose the pipeline data and processing details necessary to fully utilize all GeStore features. Finally, to reduce GeStore overhead the integration approach should not introduce a large overhead to the pipeline execution time. The above four goals are conflicting, so a good trade-off between these must be found. Below, we describe the approaches and discuss the benefits and drawbacks of each approach.

TABLE IV. Pipeline managers.

Pipeline manager	Description
GePan	High-performance computing platform scripts
Troilkatt	Data-intensive computing pipeline manager
Galaxy	Biological data analysis framework

TABLE V. Interaction approaches.

	User effort	Developer effort	Missing features	Performance
Key-value	Low	High	Low	High
External tool	High	Low	Low	High
File system	Low	Moderate	High	Medium

A. Key-value Store

We developed GeStore to provide incremental updates to the Meta-pipe (unpublished) metagenomics analysis pipeline developed and deployed at the University of Tromsø. Meta-pipe implements a custom pipeline manager called GePan that executes the analysis tools in parallel on a compute cluster using the Open Grid Engine (OGE) or Torque. GePan generates shell scripts that choose meta-databases and file format conversions based on tools and biological domains specified by the user. GePan represents pipeline managers written for a specific pipeline and a traditional HPC cluster.

GePan describes an analysis pipeline as a set of shell scripts generated from a set of parameters that specify the tools to run and their parameters. It will generate two kinds of shell scripts: 1) a set of job scripts, 2) a job submission script.

For each step in the pipeline, GePan will generate a script to run the specified tool with the correct parameters. The job script will read a task-ID and job-ID from the environment and pass it to the specified tool as appropriate. The job script will also create directories and delete intermediate files.

The job submission script is a shell script that submits jobs to the job manager. The script consists of a series of `qsub`-commands that submits each tool in the pipeline to the job manager with the appropriate dependencies.

GePan uses the key-value store interface exported by GeStore. We have modified the GePan script generation to replace file copy operations in the job scripts with GeStore calls. GeStore method arguments are set at runtime. This includes the pipeline ID, tool to execute, tool input data, and the meta-databases to use. GePan can also determine if the file retrieved is a meta-database, input data or intermediate data. In addition, GePan has information about filters to use and other user-supplied parameters. GePan can specify for which files GeStore should generate incremental versions, which files need additional parameters for increment generation, and which files are regular non-incremental files.

The integration requires very little user involvement. The user must set a "-g" parameter in GePan to specify that GeStore calls should replace file systems operations. In addition, the user may provide a filter to generate a subset of meta-database (for example for only one biological taxon).

The development effort for the integration is high, since the developer needs to have extensive knowledge of how GeStore will be used in different parts of the pipeline. All file accesses are from scripts generated by GePan, hence GePan must be modified to replace these file accesses with GeStore calls. In total, we added about 300 lines of code to the 14.000 line GePan codebase. We did not modify any Meta-pipe tools.

GeStore incurs an overhead for small files (a few megabytes), for which the time to generate incremental updates is larger than the reduction in execution time. To avoid this overhead the pipeline manager can set a file size threshold for files managed by GeStore.

The main advantage of this approach is that the incremental updates are hidden from the user, and that GePan can directly use the full feature set of GeStore. However, the integration requires more development time and a detailed knowledge of the GePan pipeline manager. This integration approach is therefore best suited for small, specialized pipeline mangers that the developer knows in detail.

B. Filesystem Interface

The Troilkatt system (section III.A) is a system for batch processing pipeline tools that analyze a large-scale compendium. Troilkatt represents specialized pipeline managers designed for data-intensive computing. It manages the setup of tool execution including the specification of input and meta-files for a tool and the management of tool output files. A tool may read and write the files directly from HDFS. Alternatively, if the file requires a POSIX file system interface, Troilkatt automatically copies the files to and from a local file system before and after tool execution. We designed Troilkatt to manage files in several file systems. It therefore provides a common file system interface. The interface was implemented for POSIX file systems and for HDFS. To integrate GeStore with Troilkatt we chose to implement the file system interface for GeStore.

To use GeStore, the user sets a field in the pipeline configuration XML file. The user does not need to have a detailed knowledge of GeStore nor Troilkatt.

The required development effort is moderate. A detailed knowledge of the internals of Troilkatt is not required to implement the interface. However, the interface is complex and the documentation may be lacking.

The main benefit of this approach is that it does not require changes to core Troilkatt code. However, there are two disadvantages. First, GeStore must infer the required information for its parameters from file-, and pathnames. These may not always have all required information, such as pipeline IDs, the type of file, or the analysis tool to execute. Second, we must parse file pathnames to automatically infer the file type and pipeline ID. If the information is ambiguous, GeStore must use the non-incremental full-file format and can therefore not provide the execution time reduction of incremental updates. This integration approach is suited for file-based pipeline mangers.

C. Galaxy Tool Wrapper

Galaxy (section II.B) is probably the most popular pipeline manager for biological data processing. Galaxy represents a big and complex pipeline manager with 165.000 lines of code. We have integrated GeStore with Galaxy by implementing a Galaxy tool wrapper for GeStore. The pipeline manager will execute a script that calls GeStore to read incremental files and write partial result files.

The user must add the GeStore tool to her pipeline, and specify parameters including a unique pipeline-ID, the file ID, incremental file retrieval, and the file type. In order to specify these parameters the user must know the GeStore API, and understand the features and limitation of GeStore. We believe this is too complicated for ordinary Galaxy users, but still useful for power users that use Galaxy to specify and tune pipelines for large datasets.

The approach is easy to implement, and does not require any changes to the large Galaxy codebase. We implemented the GeStore tool using the tool API provided by Galaxy. It is up to the user to specify the GeStore parameters. She can therefore configure GeStore to introduce minimal overhead.

The main benefit of this approach is the clear separation between GeStore and the pipeline manager, and hence no changes are required to the pipeline manager. The main disadvantage is manual configuration of GeStore. This approach is suited for complex pipeline managers where it is not possible, or practical, to modify the code.

D. Galaxy Backend Integration

To combine the benefits of pipeline specifications in the popular Galaxy tool, and the close integration of GeStore with GePan, we modified GePan to use Galaxy LWR (Light Weight Runner). LWR provides a way to run a Galaxy tool on a different computer such as a compute cluster node. The user can thereby configure a pipeline that uses GePan with GeStore without setting any GeStore arguments. The GeStore call arguments will be set at runtime and provide the same low GeStore overhead as for the key/value store integration.

In order for GePan to be run by LWR it needs to be implemented as a Galaxy tool. The tool first starts GePan, and then waits for it to complete. When the GePan tool is run from Galaxy, Galaxy will notify LWR via an AMQP message broker and GePan will run on the cluster.

In order to make the integration more reliable we developed a Python script that automates the process of inferring the status of a GePan job. The script verifies the existence of output files, `grep` log directories for errors and checks cluster status via DRMAA. The script also implements a method to wait for a GePan pipeline to complete. Our implementation of GePan currently supports Open Grid Scheduler and Torque.

The shell scripts generated by GePan are brittle and prone to errors. In addition, GePan itself does not contain sufficient logic to determine if a job has failed or succeeded; there are numerous ways for which any of the shell scripts or the tools themselves could fail. After GePan has been run from the command line the user will manually test for failure conditions by querying `qstat`, use `grep` in the log directory and manually inspect output-files. The existing Galaxy integration cannot detect certain modes of failure, such as mid-run failures.

A better solution would be to generate an intermediate representation of the pipeline and then either use that to generate a Condor workflow [29] or to submit jobs directly via DRMAA.

V. Discussion

The main question addressed by this paper is why data-intensive computing systems are typically not used for biological data processing. We provide three examples of the benefits by using such systems. Of the three systems: Troilkatt is in production use, we want to deploy GeStore to a production platform, while Mario requires more development before it is ready for production use. Here we discuss challenges for deploying such systems for production use.

Challenge 1 – Code quality. Before deploying code on a production system, it should be well tested. Most biological data processing code is developed in research projects, where

test suites are typically not implemented. We have implemented test cases for Troilkatt and GeStore, and we found these very useful when there are multiple developers, and for code run on multiple platforms.

Challenge 2 – Interdisciplinary collaboration. We believe open source code is necessary but not sufficient to ensure adaptation. All three systems described here are open source. But more importantly; we have close collaboration with groups developing and deploying analysis pipelines. Through that collaboration, we can deploy the systems in production use and thereby demonstrate the usefulness of the systems to our biological end-users.

Challenge 3- Integration. We believe it is critical that an infrastructure system can support the analysis tools already in use by the analysts. We have achieved this by integrating our systems with the popular Galaxy pipeline manager, and by ensuring that analysis tools can be used unmodified. The disadvantage of this approach is that the framework may limit the features and services that can be used. In addition, the tools may not fully utilize the power of a data-intensive analysis programming model. In our experience, integration is not straightforward and often requires in-depth knowledge of both systems. There is also a lack of standard interfaces for system integration in biological frameworks.

Challenge 4 – Cluster management. In our experience, the lack of clusters designed for data-intensive computing is an important limitation for adaptation. We have been in a position where either our collaborators or we built such clusters. But we have also ensured that our systems integrate well with traditional HPC infrastructure systems. Lately, systems such as Mesos have emerged that attempt to improve the integration of HPC systems and data-intensive computing systems.

Challenge 5 – GUI integration. Data-intensive computing systems must also be integrated with statistical analysis or visualization systems used by the end-users. Today these are often implemented as web applications that use a well-defined REST interface for data access (e.g. Kvik [44]) or are encoded as HTML documents (e.g. Krona [42]). For these it is often enough to update data files. But there are also systems such as the METAREP [43] metagenomics analysis framework that combine the Lucene [45] search engine, with the R statistical analysis framework, and a PHP based web application. Integration with such systems require in-depth knowledge of the data accesses of the visualization system.

VI. Conclusions and Future Work

We have transparently extended flexible biological data analysis frameworks to utilize data intensive computing infrastructure systems. We have integrated our data-intensive computing systems with three pipeline managers, including Galaxy. The integration required localized changes to the code, and it often utilized existing interfaces for the integration. For all cases, the analysis tools were unchanged.

We believe our work demonstrates that biological data processing can utilize data-intensive computing infrastructure systems. We focus on services that are important for biological data analysis that only require a small computer clusters. We use an approach that is transparent for biological analysis tools. We also believe our results can be applied on other data analysis fields with large-datasets and flexible analysis pipelines.

As future work, we intend to deploy our developed infrastructure systems on production systems. We are also improving our Galaxy integration, including better error handling. We are also developing several other biological data processing systems for large-scale datasets in collaboration with our biology and medicine collaborators.

Troilkatt, GeStore and Mario are all open source:

- https://github.com/larsab/troilkatt
- https://github.com/EdvardPedersen/GeStore
- http://bdps.cs.uit.no/code/mario/

Acknowledgments

Thanks to Tim Kahlke who developed GePan. Thanks to Kai Li, Olga Troyanskaya, and Alicja Tadych for their help developing and deploying Troilkatt. Thanks to our colleagues at the Tromsø Elixir node for their expertise in developing and deploying biological data processing systems.

References

[1] S. D. Kahn, “On the Future of Genomic Data,” *Science (80-.).*, vol. 331, no. 6018, pp. 728–729, Feb. 2011.

[2] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. H. Yang, and J. Zhang, “Bioconductor: open software development for computational biology and bioinformatics.,” *Genome Biol.*, vol. 5, no. 10, p. R80, Jan. 2004.

[3] D. Blankenberg, G. Von Kuster, E. Bouvier, D. Baker, E. Afgan, N. Stoler, J. Taylor, and A. Nekrutenko, “Dissemination of scientific software with Galaxy ToolShed.,” *Genome Biol.*, vol. 15, no. 2, p. 403, Jan. 2014.

[4] J. Goecks, A. Nekrutenko, and J. Taylor, “Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.,” *Genome Biol.*, vol. 11, no. 8, p. R86, Jan. 2010.

[5] 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. 3045–54, Nov. 2004.

[6] "Message Passing Interface Forum, “MPI: A Message-Passing Interface Standard. Version 3.0,” 2012.

[7] “OpenMP Application Program Interface version 4.0,” 2013. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

[8] J. Li, W.-K. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, “Parallel netCDF: A High-Performance Scientific I/O Interface,” *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, 2003.

[9] “Parallel HDF5.”, 2014, http://www.hdfgroup.org/HDF5/PHDF5/

[10] “Open Grid Scheduler.” http://gridscheduler.sourceforge.net/.

[11] “TORQUE Resource Manager - Adaptive Computing.” http://www.adaptivecomputing.com/products/open-source/torque/.

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” in *ACM SIGOPS Operating Systems Review*, 2003, vol. 37, no. 5, p. 29.

[13] J. Dean and S. Ghemawat, “MapReduce: a flexible data processing tool,” *Commun. ACM*, vol. 53, no. 1, p. 72, Jan. 2010.

[14] “Hadoop homepage,” 2014. http://hadoop.apache.org/.

[15] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “BigTable: A Distributed Storage System for Structured Data,” *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1–26, Jun. 2008.

[16] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing,” in *NSDI’12*

Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, 2012.
[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.
[18] L. A. Bongo, E. Pedersen, and M. Ernstsen, "Data-intensive computing infrastructure systems for unmodified biological data analysis pipelines," in *Eleventh International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics (accepted for publication)*, 2014.
[19] "MountableHDFS." http://wiki.apache.org/hadoop/MountableHDFS.
[20] R. C. Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics.," *BMC Bioinformatics*, vol. 11 Suppl 1, p. S1, Jan. 2010.
[21] "Apache Spark." https://spark.apache.org/.
[22] "Apache HBase." http://hbase.apache.org/.
[23] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a high-level dataflow system on top of Map-Reduce: the Pig experience," in *Proc. of the VLDB Endowment*, 2009, vol. 2, no. 2, pp. 1414–1425.
[24] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," in *Proceedings of the VLDB Endowment*, 2009, vol. 2, no. 2, pp. 1626–1629.
[25] "Cascading." http://www.cascading.org/.
[26] A. Lakshman and P. Malik, "Cassandra," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, p. 35, Apr. 2010.
[27] "Mahout homepage," 2014. https://mahout.apache.org/.
[28] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor, "Galaxy CloudMan: delivering cloud compute clusters.," *BMC Bioinformatics*, vol. 11 Suppl 1, no. Suppl 12, p. S4, Jan. 2010.
[29] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-a hunter of idle workstations," in *Proceedings. The 8th International Conference on Distributed*, pp. 104–111. 1988
[30] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: a platform for fine-grained resource sharing in the data center," in *NSDI'11 Proceedings of the 8th USENIX conference on Networked systems design and implementation*, 2011.
[31] "Docker." https://www.docker.com/.
[32] M. Abouelhoda, S. A. Issa, and M. Ghanem, "Tavaxy: integrating Taverna and Galaxy workflows with cloud computing support.," *BMC Bioinformatics*, vol. 13, no. 1, p. 77, Jan. 2012.
[33] "Public Data Sets on AWS." http://aws.amazon.com/public-data-sets/.
[34] "Services at EMBL-EBI; Embassy Cloud." http://www.ebi.ac.uk/services.
[35] C. Y. Park, A. K. Wong, C. S. Greene, J. Rowland, Y. Guan, L. A. Bongo, and O. G. Troyanskaya, "Functional Knowledge Transfer for High-accuracy Prediction of Under-studied Biological Processes," *PLoS Comput. Biol.*, vol. 9, no. 3, p. e1002957, Mar. 2013.
[36] A. K. Wong, C. Y. Park, C. S. Greene, L. A. Bongo, Y. Guan, and O. G. Troyanskaya, "IMP: a multi-species functional genomics portal for integration, visualization and prediction of protein functions and networks.," *Nucleic Acids Res.*, vol. 40, no. Web Server issue, pp. W484–90, Jul. 2012.
[37] T. Barrett, D. B. Troup, S. E. Wilhite, P. Ledoux, C. Evangelista, I. F. Kim, M. Tomashevsky, K. A. Marshall, K. H. Phillippy, P. M. Sherman, R. N. Muertter, M. Holko, O. Ayanbule, A. Yefanov, and A. Soboleva, "NCBI GEO: archive for functional genomics data sets--10 years on.," *Nucleic Acids Res.*, vol. 39, no. Database issue, pp. D1005–10, Nov. 2010.
[38] E. Pedersen, N. P. Willassen, and L. A. Bongo, "Transparent incremental updates for Genomics Data Analysis Pipelines," in *HiBB 2013 – 4th Workshop on High Performance Bioinformatics and Biomedicine. LNCS, vol. 8374*, 2014.
[39] M. Magrane and U. Consortium, "UniProt Knowledgebase: a hub of integrated protein data.," *Database (Oxford).*, vol. 2011, p. bar009, Jan. 2011.
[40] M. Ernstsen, E. Kjærner-Semb, N. P. Willassen, and L. A. Bongo, "Mario: interactive tuning of biological analysis pipelines using iterative processing," in *in Proc. of 5th International Workshop on High Performance Bioinformatics and Biomedicine. LNCS, vol. 8805*, 2014.
[41] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP '13*, 2013, pp. 423–438.
[42] B. D. Ondov, N. H. Bergman, and A. M. Phillippy, "Interactive metagenomic visualization in a Web browser.," *BMC Bioinformatics*, vol. 12, no. 1, p. 385, Jan. 2011.
[43] J. Goll, D. Rusch, D. M. Tanenbaum, M. Thiagarajan, K. Li, B. A. Methé, and S. Yooseph, "METAREP: JCVI Metagenomics Reports - an open source tool for high-performance comparative metagenomics.," *Bioinformatics*, vol. 26, no. 20, pp. 2631–2, Aug. 2010.
[44] B. Fjukstad, K. S. Olsen, M. Jareid, E. Lund, and L. A. Bongo, "Kvik: Interactive exploration of genomic data from the NOWAC postgenome biobank," in *Proc. of NIK 2014*, 2014.
[45] "Apache Lucene." http://lucene.apache.org/.