2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing

Towards a MOLGENIS based computational framework

Heorhiy Byelas*, Alexandros Kanterakis† and Morris Swertz‡
*University of Groningen, Netherlands
Email: h.v.byelas@rug.nl
†University Medical Centre Groningen, Netherlands
Email: a.kanterakis@rug.nl
‡Groningen Bioinformatics Centre, Netherlands
Email: m.a.swertz@rug.nl

***Abstract*—High-throughput bioinformatics research is complex and requires the combination of multiple experimental approaches each producing large amounts of diverse data. The analysis and evaluation of these data are equally complex requiring specific integrations of various software components into complex workflows. The challenge is to provide less technically involved bioinformaticians with simple interfaces to specify the workflow of commands they need while at the same time scale up to hundreds of jobs to get the terabytes of genetic data processed by recent methods.**

Here, we present a computational framework for bioinformatics which enables data and workflow management in a distributed computational environment. Firstly, we propose a new data model to specify workflow execution logic on available network resources and components. Our model extends existing generic workflow and bioinformatics models to describe workflows compactly and unambiguously. Secondly, we present the implementation of our computational framework, which is constructed as a computational cloud for bioinformatics using open source off-the-shelf components. Finally, we demonstrate applications of the framework on complex real-world bioinformatics tasks.

***Keywords*-Bioinformatics, computational cloud, workflow management system**

I. Introduction

Workflow management systems are widely used in bioinformatics [1], [2], [3]. They describe computational steps of analyses. Every step of the workflow can be an individual operation or a set of thereof, which can be executed under particular conditions. To deal with large data sets processed in these workflows, computational operations and/or data are often distributed across available computational resources and executed in parallel. For bioinformaticians, a familiar example of a single operation is a command-line invocation of an analysis tool (*e.g.* plink, Beagle [4], [5]), which performs the analysis on genotype/phenotype data. A manual invocation of tools can be time consuming and error-prone, especially, if it should be repeated many times per day for different data sets and with different command-line parameters (*e.g.* 12 analysis steps for 710 samples).

Besides computational steps, many other elements of workflows should be specified. These elements include data on which computations are performed, tools which take part in workflows and users who perform computations. Various bioinformatics warehouses (*e.g.* Biowarehouse, Genoquery [6], [7]) were constructed to manage heterogeneous workflow elements. One of the main challenges in constructing a data warehouse is to keep its functionality not too broadly defined. Otherwise, construction can take much more efforts and time to implement than available. Still, the constructed system should provide benefits promised and, in a bioinformatics context, that is an automated user access to available bioinformatics workflows, tools and data.

In this paper, we present a MOLGENIS-based Computational Framework (MCF) for bioinformatics. The framework supports distributed workflow execution initialized from data warehouses, which are generated using MOLGENIS [8]. The main purpose of combining data and workflow management is to enable customization of data warehouses for a particular bioinformatics research, keeping their functionality as simple as possible. Additionally, the computational warehouse back-end, as it is implemented now, can be easily replaced by alternative computational infrastructures in the future.

The paper is structured as follows. Section II reviews related work in workflow management systems for bioinformatics. Section III presents the data model used in the system and explains decisions made. Section IV presents the framework design and implementation. Section V presents several applications of our system on real-world bioinformatics problems. Section VI discusses advantages and limitations of the framework. Section VII concludes this paper and outlines potential directions for future work.

II. Related Work

The aim of this paper, introduced in Section I, is to construct a data analysis framework, which combines computational and data management for bioinformatics in a single solution. Furthermore, we would like to perform our computations in a distributed execution environment. Hence, this section contains related work on workflow management systems which are focusing on the bioinformatics domain.

An extensive review and taxonomy of generic grid enabled workflow management systems was done by J. Yu *etc.* [9] in 2005. Some of the mentioned generic management

1066-6192/11 $26.00 © 2011 IEEE
DOI 10.1109/PDP.2011.53

systems, such as Taverna [1] or Kepler [10], are now widely used in the bioinformatics domain. Taverna is a suite of tools to design and executes workflows. It allows users to integrate third-party software tools, which are described as web services, into workflows. A set of services is not fixed and new tools can be added by users. Workflows are presented as graphs constructed using the Taverna visual language. Additionally, Taverna allows users to monitor workflow execution and examine the provenance of the data described using Open Provenance Model [11]. Combining Taverna with RAPID [12] allows to submit workflow jobs to the computational grid. This is done by declaring jobs in the RAPID specification file. How jobs are defined depends on the used grid system.

Besides generic workflow management systems specialised for life-sciences, specific bioinformatics management systems, such as Galaxy [2], have recently been developed. Galaxy is a web environment, where users can create pipelines by combining a large variety of bioinformatics tools. Each tool is thoroughly documented in the rich XML-based language. Data can be uploaded, preprocessed and visualised through the same environment, which includes mechanisms for interactive workflow execution and history logging. It is targeted at genome-wide scale analysis mainly for sequences, alignments and functional annotations. Biopipe [13] can be considered as an alternative to Galaxy. It offers a set of wrappers that build a common interface for accessing existing bioinformatics tools and data.

Still, combining data and computation management in a single system for bioinformatics, that enables data provenance recording, is lacking. If compared with the work listed above, we are constructing a Galaxy-like environment, where jobs are specified in a RAPID-like fashion for executing in a computational cloud. The main aim of our workflow language is to efficiently map and execute jobs on available resources. This is discussed in detail next.

III. Data Model

The main goal of our model is to unambiguously specify a bioinformatics workflow execution in a distributed computational environment. The model includes the entire workflow specification and specifications for individual workflow tasks. Hence, we divide our model into two structural layers or interfaces. These are a *job* interface to specify a workflow and a *tool* interface to specify an individual workflow task. These interfaces are used in two main framework use scenarios:

- specifying a workflow or job for execution, and
- adding a new tool to the framework tool repository. Hence, analyses provided by the tool can be included into workflows.

In our model, we are interested to specify any external analysis tool, which can be invoked from a command line or be run as an executable script (*e.g.* a shell or R-script [14]).

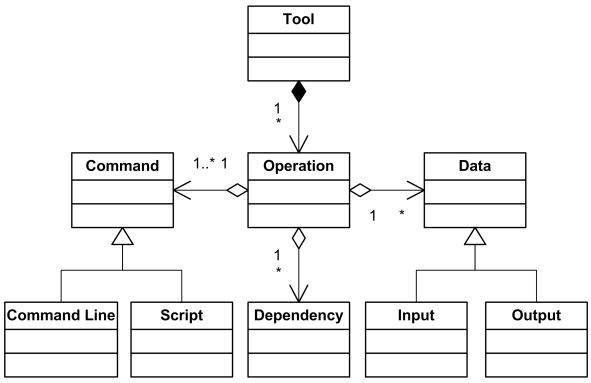

Figure 1. Tool model

The *tool* model, shown in Figure 1, is similar to the Galaxy tool file [2], where a tool is described as a set of operations, which it can perform. A tool operation can participate in workflow execution. In contrast with the Galaxy description, we do not have complex operation parameters in the model. Instead, we treat an operation with different parameters as a set of separate operations. A list of tool operations grows depending on the complexity of the operations parameterisation. Still, the specification of an individual operation remains simple.

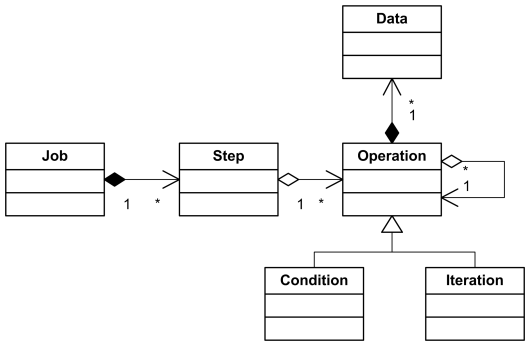

Figure 2. Job model

There are a number of reasons to keep the description of a single operation as simple as possible. Firstly, bioinformaticians typically use only a subset of available tool features in the majority of analysis workflows. Moving towards simpler operations importantly lowers the bar for bench biologists to be comfortable running operations without the need of massive bioinformatics background knowledge. Secondly, it is possible that a tool will be added to the framework by a person, who is not an expert in all operation parameters, but only interested in a particular execution scenario. In this case, it would be easy to add a short operation specification than an exhaustive list of all operation parameters. Furthermore, additional tool parameters can be added later as a new operation. All parameters defined in the tool can be reused by new operation definitions in good spirit of the Don't Repeat Yourself (DRY) principle. Inputs, outputs and

dependencies of a tool operation are specified explicitly to assure the correctness of workflow and provide means for dynamic error handling during workflow execution.

The second part of our model is the *job* model (see Fig. 2). It describes a workflow as a sequence of steps. A *step* consists of a set of operations. Operations of one step are executed in parallel and do not depend on each other. An *operation* can be a *condition* to specify that this operation should be executed under some conditions, which are results of the previous workflow step. The *condition* object class is present in the model, but it was not used, so far, in the designed workflows (Sec. V). Additionally, an *operation* can be an *iteration* to specify repeating of the same operation for different data (*e.g.* repeat the same operation for all chromosomes).

In the *job* model (Figure 2), an *operation* object is a concrete instance of a tool operation from the *tool* model (Figure 1). Our description matches three first stages of the RAPID [12] job specification. It specifies (1) data (*e.g.* files, libraries, executables) to be transferred to the host where the computation will take place, (2) command-lines to be executed, and (3) data to be transferred back.

We would like to treat our computational environment as a cloud. Computational clouds are typically divided on segments. Transferring large amounts of data between cloud segments can easily become a bottleneck of a workflow execution. Hence, computations should be performed close to the data storage. However, as practice shows, transferring gigabytes of data from a single machine to a computational cluster is not a problem if one intends to run computationally intensive analyses for several days. An example of a long-duration workflow is given in Section V-A.

IV. MCF DESIGN AND IMPLEMENTATION

We started our domain analysis with considering several business use cases and trying to derive conceptual ones which cover all use cases to some extent. The list of our business use cases includes genotyping and quality checking [15], imputations on genetic data [16], SNP association analysis [17] *etc.* We try to identify a sequence of actions conducted by a final system user, *i.e.* a biologist or bioinformatician, to carry out his research.

We end up with four conceptual user steps of a biologist (researcher):

1) Log into the system
    a) Log in as a researcher
    b) See data available to you
2) Select a workflow
    a) Select data
    b) Select workflow parameters
3) Run a workflow and see its execution status
4) View results
    a) Explore results
    b) Download results, if you have permission

Besides a biologist, a bioinformatician (*i.e.* framework administrator) is another actor in our business use-cases. He is responsible to maintain the system, that is mainly add or update data and tools in the framework. His actions look as follows:

1) Log into the system
    a) Log in as an administrator
    b) See available data/tools
2) Add new data/tools
    a) Set up new data/tools
    b) Register new data/tools in the system database
3) Add a new workflow
    a) Design a new workflow
    b) Register a new workflow in the system database

We identified four user interfaces to support listed above actions:

- I1: Job input interface, where a workflow can be selected for execution,
- I2: Data interface, where results of the workflow are shown,
- I3: Tool deployment interface, where a new tool can be specified and uploaded,
- I4: Data deployment interface, where a new data can be specified and uploaded.

All user interfaces are HTML-based interfaces generated using MOLGENIS. They build the User Interface layer of the framework.

The MCF architecture consists of three layers as it is shown in Figure 3. These layers are:

- User interface layer,
- MCF Applicational layer, which contains the framework logic, and
- Infrastructure layer, which contains third-party components for data and computational infrastructure.

The MCF Applicational Layer combines computational and data management in a single solution. This is archived by adding the computational back-end package (*Compute Manager*) [18] to the data warehouse generated using MOLGENIS. Two subsystems communicate via a narrow interface containing several methods. The core interface methods are:

- `void setJob(Job newJob)`
- `Job getJob(int ID)`

MOLGENIS provides the data management interfaces with simple 'CRUD' operations to track and trace metadata (*i.e.* protocols, samples, experiments, samples), file attachments to manage raw, intermediate and result data, and domain specific extensions like 'data matrix' operations to help bioinformaticians to work with their data. *Compute Manager* provides functionalities of a workflow and resource manager of the computational cloud. Let us consider these functionalities in detail.

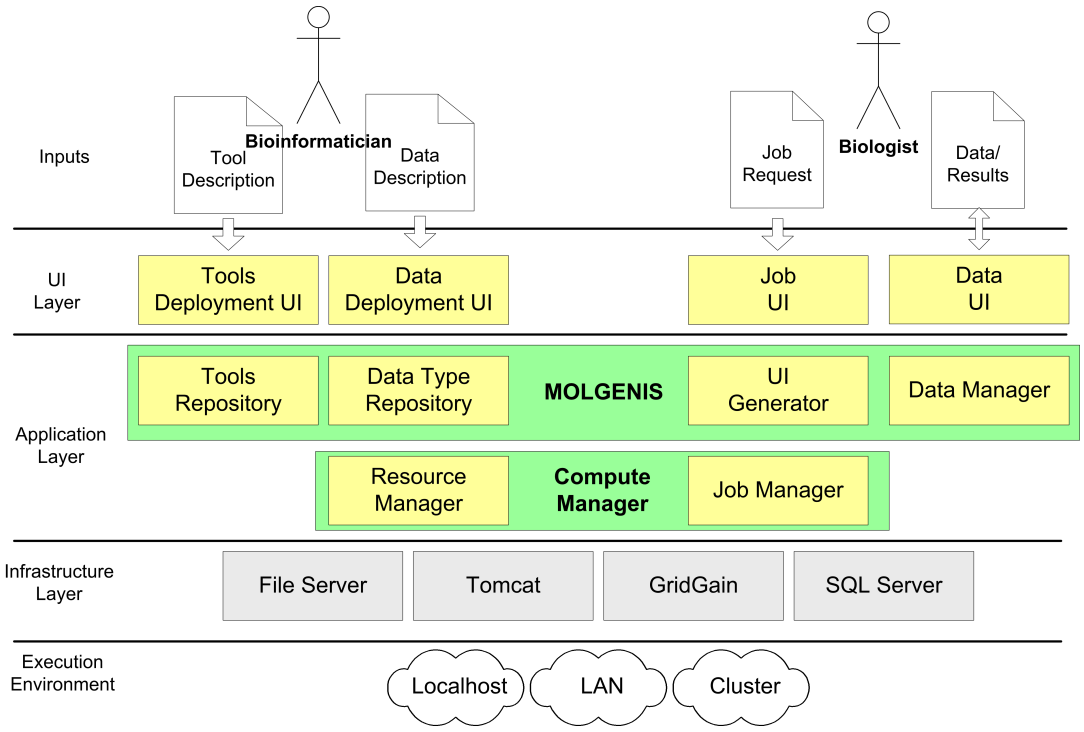

Figure 3. Layered MCF architecture

A. Data management

We built our data management on the MOLGENIS system [8]. MOLGENIS is a model-driven system that auto-generates from a data model described in XML to a fully functional data platform including database back-ends, web user interfaces for biologists and programmatic interfaces for bioinformaticians. An example of the generated user interface is shown in Figure 4.

MOLGENIS either generates a high-performance 'server' edition, which requires software installation on a server or cluster, or a limited 'standalone' edition that runs on a desktop computer without any additional configuration. Both usages are shown in Section V. The database layer is generated as SQL files with database `CREATE` statements that are loaded into either MySQL (server), PostgreSQL (server) or HSQLDB (standalone). The API layer is generated as Java files either served via Tomcat (server) or Jetty (standalone). A Java class is generated for each data type (*e.g.* a class *Gene*). All data can be queried programmatically via a central *Database* class. For example, the command `db.find(Gene.class)` returns all *Gene* objects in the database.

To enhance the performance of data loading in MOLGENIS, the 'batched' update methods of Java DataBase Connectivity (JDBC) package and 'multi-row-syntax' of MySQL are used. This allows to insert 10,000s of data entries in a single command, that is 5 to 15 times quicker

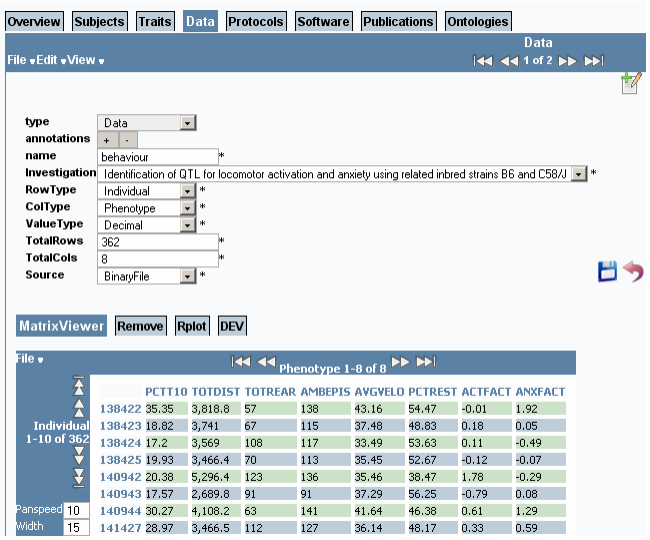

Figure 4. An example of the generated user web interface (a part of the screen)

than standard one-by-one updates and allows data loading of 20k records/second on commodity desktop machines. The Java/API is exposed with a SOAP/API, REST/API and R/API, so MOLGENIS can also be accessed via web service tools like Taverna, REST or R, respectively (accessible via hyperlinks in the GUI).

Existing databases can be quickly enriched with a MOL-

GENIS front-end by generating data management infrastructure from their models. The standard generated platform can be extended via plug-ins to integrate research specific processing protocols. Obviously, the generator can be re-run often to build a new data warehouse for a new research. All this provides the database API we use as a data management system for *Compute Manager*, most importantly the REST system for retrieving and putting data sets via the data deployment interface (I2 and I4).

We enhanced MOLGENIS with two new modules for this project:

- We loaded our tool model inside MOLGENIS to generate the tool deployment interface (I3). Hence, end users can configure new tools, their parameters and upload binaries.
- We added a user interface plug-in to MOLGENIS to generate the job input interface (I1) for workflow enactments and monitoring of job completion, and data interface (I2) for viewing results.

Now, we have all four required user interfaces and the data management part of the system.

B. Computational management

Our computational infrastructure is organised as a "cloud" and implemented using the GridGain 2.1.1 development platform [19]. The whole computational logic is located at one cloud node, which is the *MCF Compute Manager* node. The rest of the computational base is a standard GridGain software deployed in a local network, cluster or server. MOLGENIS data management modules, described in Section IV-A, are also deployed on the *MCF Compute Manager* node. The topology of our "cloud" is shown in Figure 5.

Compute Manager [18] consists of two modules:

- *Job Manager*, which distributes *jobs* across *Worker* nodes and monitors their executions, and
- *Resource Manager*, which starts and stops *Worker* nodes on the cluster.

The *Job Manager* logic is rather straightforward and can be easily adjusted for use in a specific cluster or server. After a *job* object (see Section III) is received by *Job Manager*, it is registered in the database and passed to the *Worker* nodes for execution. There are two different kinds of *Worker* nodes in the system. These are *Resident Workers* and *Extra Workers*. Basically, these nodes are the same standard GridGain nodes and differ only by name or a cloud segment.

Why do we need two different kinds of nodes in the system, if these nodes have the same functionality? As we mentioned in Section I, a workflow operation is an execution of a bioinformatics analysis tool, which is invoked from a command line. A usual output is files and a standard command-line output or/and error. The difference between two kinds of *Worker* nodes is in a way analysis tools are invoked from them. *Resident Worker* starts a job by submitting a shell script to a cluster job scheduler. In contrast to *Resident Worker*, *Extra Worker* directly invokes an analysis tool. A cluster scheduler can be circumvented in this way.

Extra Workers are pre-started and stopped by *Resident Worker*. *Resident Worker* receives a command from *Resource Manager* and starts *Extra Workers* by submitting a script to the cluster scheduler to start them. After being started, *Extra Workers* communicate to *Job Manager* and register themselves. In practice, it can take more time to pre-start many *Extra Workers* for direct parallel execution of analysis operations than submit scripts to a cluster scheduler to execute the same operations. Furthermore, running many *Extra Workers* in the system increases the network load on the *Job Manager* node. Still, *Extra Workers* can be efficiently used in the system having an advanced strategy to pre-start them, that is planned to be developed in the future.

Resource Manager is required only if a computational cluster is used in the system. Its logic is also straightforward and directly depends on the policies of the cluster used. We tested our framework on the Millipede HPC cluster [20], which appears in the TOP500 supercomputers list [21]. This cluster has a policy that any cluster job execution should not exceed the ten days limit to assure availability of cluster resources to all users. This means, that *Resident Worker* cannot run longer that ten days either. In our current implementation to keep a cluster as a part of our computational cloud, *Resident Worker* starts a new *Resident Worker* node in some time before it will be removed by the cluster administrator, *e.g.* two days before the end of a ten-days period. The script for starting a new *Resident Worker* is submitted to the cluster scheduler and processed in some time depending on a cluster load. Hence, we assure that at least one *Resident Worker* is running on the cluster.

All information about execution (*i.e.* STARTED, ERROR or DONE statuses of operations with timestamps) are saved in the MOLGENIS database. Typically, files produced by workflow operations are not sent to the *Compute Manager* node and stay in a cluster or server. Therefore, they can be used in other workflows and downloaded later by a separate operation. If *Workers* in the cluster share one data and environment space, *Worker* nodes in a local network have their individual execution spaces. This means that data for analysis should be transferred to these individual nodes, tools and environment variable should be set-up before the actual analysis will take place. Currently, many administrative operations, such as deletion of unused files, setting up environment variables, *etc.*, are done manually in the system.

Now, we include the Millipede HPC cluster and several individual machines in our cloud implementation. However, public computational clouds can also be added to the framework. We tested our solution on Amazon EC2 [22], for

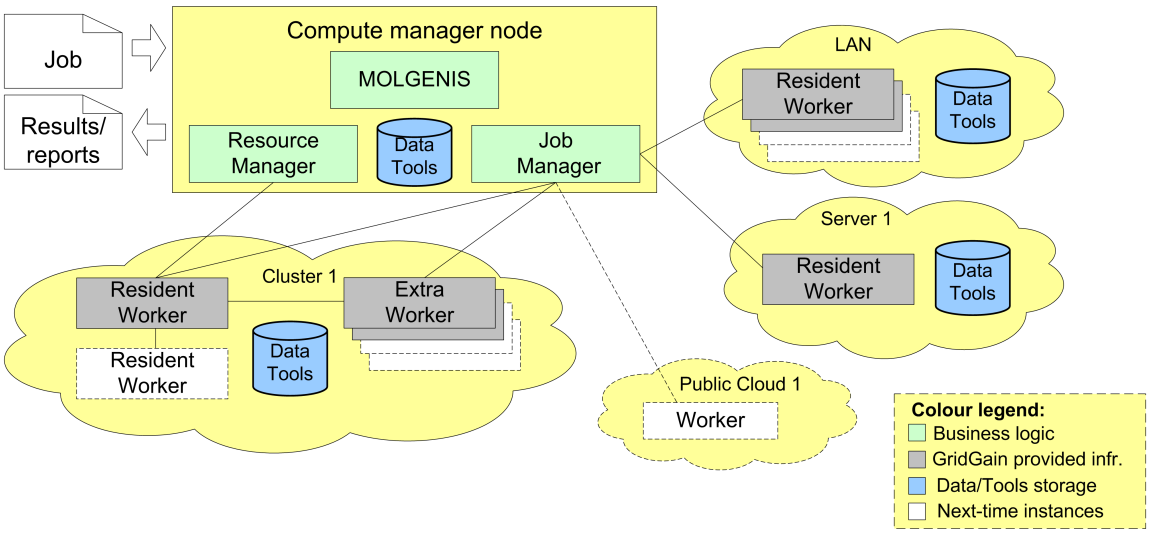

Figure 5. The computational framework topology

which GridGain provides an image. Consequently, *Worker* nodes can be easily started on the Amazon EC2. Additionally, our computational back-end, which is built on GridGain, can be replaced by another cloud or grid computational infrastructure, like *e.g.* DPU [23], or support several of them.

In our framework, we can specify any pipeline consisting of command lines or scripts. However, due to the nature of workflows we considered so far, our system intended to execute *jobs* which last from seconds and minutes to hours and days, rather than *jobs* of milliseconds. Consequently, we do not address an issue of a heavy network load on *Compute manager* caused by messages with status information received from *Worker* nodes. *Job* examples are discussed in more detail further.

V. Applications

To assess the feasibility and effectiveness of the proposed solution, we conducted a number of studies on real-world bioinformatics workflows. In this paper, we present two dissimilar by computational intensiveness bioinformatics analyses. These are an imputation of genetic data (Sec. V-A) and SNP association analysis (Sec. V-B). Both of them are regular analyses on genotype data, however, they vary in a size of data for analysis and computational time. If the imputation pipeline should be performed on the cluster, the SNP association analysis can be performed on a local machine as well.

A. Imputation pipeline

Our current imputation pipeline consists of 8 steps. Initial files to be imputed are present in the plink binary format [4]. The result of the pipeline are imputed datasets in the same format. Hence, biologists can continue to analyse imputed data without any additional preparations. The actual imputation is done using BEAGLE 3.0 [5]. The files are pre-processed in the pipeline by the in-house developed software for the TriTyper algorithm [17] and linkage2beagle [5] tools. We run imputation on large datasets of 1000 - 10000 individuals with a reference panel of 90 phased individuals (CEU HAPMap). Much larger datasets ($\approx$ 50000 individuals) are requested to be processed in the near future.

The datasets are split during pre-processing on batches of 300 individuals to parallelize imputation execution without losing the quality. To our knowledge, datasets of 300 individuals can be imputed with a decent quality using BEAGLE. All imputation batch jobs are generated in MOLGENIS and submitted to the cluster scheduler as scripts by *Resident Worker* (Sec. IV-B). The average time to impute a batch of 300 individuals is about 3 hours on the Millipede cluster. The whole current pipeline execution for dataset of several thousand individuals takes about 1-3 days depending on the cluster load.

Initially, we run the imputation workflow in parallel on 22 *Extra Worker* nodes, where every *Worker* performs imputation on a single chromosome without splitting it on batches. The execution time of the whole workflow was equal to the imputation time of the largest chromosome, which is Chr1. The measured imputation time for the dataset of 3000 individuals on the Millipede cluster was about 140 hours. The imputation time for the dataset of 10000 exceeds the cluster job execution limit of 10 days. Therefore, the dataset of 10000 individuals cannot be processed without splitting into smaller parts.

Besides the execution time to run imputation, there are disk storage and memory requirements. The imputed dataset of 3000 individuals occupies about of 35 GB of the disk space and temporary files during the pipeline execution grow

up to 100 GB. In our scenario, the whole input and output data for this workflow stay on the cluster to make them available for further analyses. Consequently, the data storage issue should be taken into account if one likes to work with many datasets. Additionally, memory swapping on the cluster enormously increases the execution time. Imputing our datasets using BEAGLE requires up to 8 GB of RAM to avoid memory swapping. Millipede cluster nodes have enough memory to avoid it, however, we had problems when running this pipeline on some other clusters.

Overall, our conclusion is that large datasets should be split into batches for imputation, even when we run analysis on powerful machines. Datasets of about 3000 individuals are feasible to process without splitting on the Millipede cluster. However, there is a probability that a process with a duration of several days would fail because of some communication errors in the cluster or network. Imputing large datasets with our pipeline on a personal computer is not feasible so far.

B. SNP association analysis

In our second analysis example, we compare running analysis on a local machine versus running it remotely on the cluster. As it is shown in Section V-A, in practice some bioinformatics analyses can be performed only on the cluster. SNP association analysis can be performed on a regular laptop having a MOLGENIS generated database (Sec. IV-A) installed on it. However, how useful and efficient is it? To answer this question, we run analysis locally having 1 *Resident Worker* (see Sec. IV-B) and remotely having 22 *Extra Workers* on the cluster. Running 22 *Extra Workers* for such analysis is not practical. However, we would like to compare executions assuming the ideal conditions on the cluster and locally.

In our scenario, a biologist selects a group of people of a certain age (*e.g.* $>$ 40 years old) with a higher than a normal blood pressure (*e.g.* $\geq$ 140/100 mmHg) as a case study. Another group of people of the same age, but with a normal blood pressure (*e.g.* $<$ 140/100 mmHg) is selected as a control group. These two datasets containing phenotype information are generated from a 'standalone' MOLGENIS database. We assume that files with genotype data for selected people are present on the disk. We run a SNP association analysis using plink [4]. In this example, the whole analysis pipeline consists of only one plink command, where command line parameters are also generated in MOLGENIS. Later, a user can review results of analysis as a graph on a generated UI web page.

Running association analysis with plink on a dataset of 500 individuals takes in average 3 seconds for one chromosome on a local machine (Intel Core 2 Duo running on 2.26 GHz with 4 GB RAM). Consequently, running this analysis for 22 chromosomes takes about 1 minute. Running the same analysis for one chromosome on the cluster took us from 20 to 50 seconds depending on the cluster load, where the actual plink execution took only 2 seconds. The rest of the execution time was spent on invoking plink by *Worker* on a cluster node and reporting results back. Here, the analysis was run in parallel on 22 cluster nodes. Consequently, the total execution time is also about 1 minute.

Overall, this analysis can be performed both on a cluster and local machine. The execution durations were nearly the same. To sum up, using a cluster to run this analysis makes sense only if a user would like to repeat it with different parameters for many datasets.

VI. Discussion

In the beginning of our work, we identified the main requirements for the future system. As such, the following discussion of our findings is also structured along these requirements.

Workflow management: In our framework, we combine data, tools, resources and workflow management in a single system. We are focusing on solving specific bioinformatics tasks. Therefore, a number of tools and data formats included into our framework is limited. We expect that adding new tools and data types will not change our data model significantly.

Performance and scalability: We included the millipede cluster [20] and local network computers into our resource infrastructure. We perform computationally intensive tasks on the cluster. An execution on the cluster is hidden from the user, who starts a workflow from the HTML-based MOLGENIS interface. At this point of time, these resources are sufficient to run our analyses. Public computational clouds, such as Amazon EC2, can be included into the system in addition. Also, we are considering to include the dutch e-science grid [24] as an alternative computational back-end.

Portability: MOLGENIS consists of industry standard Java, MySql and Apache Tomcat technology. We use virtual machines to enable easy deployment on other nodes than our *Compute manager* node (see Section IV-B) if needed.

Technical issues in real-world settings: Executing workflows in a distributed environment brings an element of uncertainty into a workflow's successful completion. The correct process termination does not ensure, that analysis results are correct and present in the system. Therefore, every workflow operation should contain flags which indicate its correct completion. These flags are very operation-dependent (*e.g.* a list of all output files and their expected sizes). A second important issue is safety of bioinformatics data. Some personal data should stay inside the institution walls and, therefore, can be processed only there. Finally, using third party tools in workflows limits data and processes distribution. The maximum execution performance can be achieved if data processing algorithms embed data and

process distribution over resources in the way it is done *e.g.* in the FPGA programming.

VII. Conclusions

We have successfully demonstrated a model that makes it achievable for non-expert bioinformaticians to deploy and run compute jobs in a cloud. We have integrated this model with existing biology data infrastructure MOLGENIS to enable large scale data management, as well as, biologist friendly user interfaces to explore data and results, and track and trace computations. Finally, we have successfully evaluated our new framework in exemplar biological applications of genotype imputation and genome-wide phenotype association studies.

We would next like to integrate the even larger computations of the next-generation proteomics and sequencing technologies, where we can profile all DNA of individuals and then analyse genomic variation in 20 days. A second development direction is to introduce advanced scheduling and failed jobs resubmission algorithms into the framework functionality.

Acknowledgment

We thank all our collaborators at the Dept. of Genetics, University Medical Centre Groningen, in particular, we thank Harm-Jan Westra and Lude Franke for developing the TriTyper tool; Groningen Bioinformatics Centre, University of Groningen; Netherlands Bioinformatics Center (NBIC), BioAssist task forces for biobanking, sequencing and proteomics and the Netherlands Proteomics Centre. Heorhiy Byelas was supported by the Netherlands Proteomics Centre (NPC-GM-WP1), Alexandros Kanterakis was supported by the University of Groningen Ubbo Emmius Fund, and Morris Swertz was supported by NWO (Rubicon Grant 825.09.008).

References

[1] T. Oinn and M. Greenwood, "Taverna: lessons in creating a workflow environment for the life sciences," *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE*, vol. 18, pp. 1067 – 1100, 2005.

[2] T. J. Blankenberg D, "A framework for collaborative analysis of encode data: making large-scale analyses biologist-friendly," *Genome Res.*, vol. 17, pp. 960 – 4, 2007.

[3] BioBike, "Biological integrated knowledge environment," 2010, http://biobike.csbc.vcu.edu/.

[4] N. B. Purcell S, "Plink: a toolset for whole-genome association and population-based linkage analysis," *American Journal of Human Genetics*, vol. 81, http://pngu.mgh.harvard.edu/purcell/plink/.

[5] Y. Z. Browning B, "Simultaneous genotype calling and haplotype phase inference improves genotype accuracy and reduces false positive associations for genome-wide association studies," *The American Journal of Human Genetics*, vol. 85, pp. 847–861.

[6] Y. P. Thomas J Lee, "Biowarehouse: a bioinformatics database warehouse toolkit," *BMC Bioinformatics*, 2006.

[7] C. F. F. Lemoine, B. Labedan, "Genoquery: a new querying module for functional annotation in a genomic warehouse," *BMC Bioinformatics*, 2008.

[8] R. J. M.Swertz, "Beyond standardization: dynamic software infrastructures for systems biology," *Nature Reviews Genetics*, pp. 235–43, 2007.

[9] R. B. J. Yu, "A taxonomy of scientific workflow systems for grid computing," *ACM SIGMOD Record*, 2005.

[10] I. Altintas and C. Berkley, "Kepler: Towards a grid-enabled system for scientific workflows," in *in proceedings of GGF10-The Tenth Global Grid Forum*, 2004.

[11] OPM, "Open provenance model," 2010, http://openprovenance.org/.

[12] T. A. Koetsier J., "Rapid chemistry portals through engaging researchers," in *Fifth IEEE International Conference on e-Science*, 2009, pp. 284–291.

[13] R. K. Hoon S., "Biopipe: a flexible framework for protocol-based bioinformatics analysis," *Genome Res*, pp. 1904–1915, 2003.

[14] M. J. CRAWLEY, *The R book*. Wiley, 2007.

[15] S. M. Xiao Y., "A multi-array multi-snp genotyping algorithm for affymetrix snp microarrays," *Bioinformatics*, pp. 1459–67, 2007.

[16] B. S. Browning B.L., "A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals," *Am J Hum Genet*, pp. 210–23, 2009.

[17] W. C. Franke L., "Detection, imputation, and association analysis of small deletions and null alleles on oligonucleotide arrays," *Am J Hum Genet*, pp. 1316–33, 2008.

[18] MOLGENIS team, "Compute framework," 2010, http://www.molgenis.org/wiki/ComputeStart.

[19] N. Ivanov, "Cloud development platform," 2010, http://gridgain.com/.

[20] Millipede, "Clustervision opteron cluster," 2010, http://www.rug.nl/cit/hpcv/faciliteiten/index.

[21] H. Meuer, "The top500 project," 2010, http://www.top500.org/.

[22] Amazon, "Elastic compute cloud (amazon ec2)," 2010, http://aws.amazon.com/ec2/.

[23] Astro-Wise, "Distributed processing unit," 2010, http://www.astro-wise.org/.

[24] BIG Grid, "the dutch e-science grid," 2010, http://www.biggrid.nl.