Scaling bio-analyses from computational clusters to grids

Heorhiy Byelas, Martijn Dijkstra, Pieter Neerincx, Freerk van Dijk,
Alexandros Kanterakis, Patrick Deelen, Morris Swertz
Genomics Coordination Center, Department of Genetics
University Medical Center Groningen
University of Groningen, The Netherlands
Email: h.v.byelas@med.umcg.nl, m.a.swertz@rug.nl

***Abstract*—Life sciences have moved rapidly into big data thanks to new parallel methods for gene expression, genome-wide association, proteomics and whole genome DNA sequencing. The scale of these methods is growing faster than predicted by Moores law. This has introduced new challenges and needs for methods for specifying computation protocols for *e.g.* Next-Generation Sequencing (NGS) and genome-wide association study (GWAS) imputation analyses and running these on a large scale is a complicated task, due to the many steps involved, long runtimes, heterogeneous computational resources and large files. The process becomes error-prone when dealing with hundreds of samples, such as in genomic analysis facilities, if it is performed without an integrated workflow framework and data management system. From recent projects we learnt that bioinformaticians do not want to invest much time in learning advanced grid or cluster scheduling tools, preferring to concentrate on their analyses, be closer to old-fashion shell scripts that they can fully control and have some automatic mechanisms taking care of all submission and monitoring details. We present a lightweight workflow declaration and execution system to address these needs, built on top of the MOLGENIS framework for data tracking. We describe lessons learnt when scaling running NGS and imputation analyses from computational clusters to grids and show application of our solution, in particular, in the nation-wide "Genome of the Netherlands" project (GoNL, 700TB of data and about 200.000 computing hours)**

I. Introduction

High-throughput analysis methods have created exciting new possibilities for unraveling genotype-to-phenotype relationships. However, these experiments are heavily dependent on large computational analysis for their success. For instance, next generation sequencing analyses typically involve about 30 computational steps such as alignment, duplicate marking, single-nucleotide polymorphism (SNP) calling, annotation and many re-indexing and quality control steps [1]. Similarly, GWAS data typically requires batching for imputation. Genomic analysis facilities typically face running many different versions of such computational pipelines on hundreds or thousands of samples. This quickly becomes a nightmare of data file logistics (raw, intermediate, result, quality controls and log data) and a computational scheduling nightmare (large, small, short, long jobs that may have error states and different computational back-ends). Furthermore, different cluster and grid middleware do not provide all necessary operations to execute bio-workflows. To address these challenges we present a practical software system that combines computational and data management for routine running of large bioinformatics analyses.

In this work, our goal is to run different workflows in a unified way and make workflow adaptation to different back-ends, such as clusters and grids environments more standard and easier. Hence, users can choose a computational back-end with less load to run analyses, run workflows in different back-end environments and achieve identical analysis results and, if it is needed, combine results together. We do not want to add another middleware layer above the back-end services, such as job schedulers, but prefer to minimize an overhead to run shell scripts in different environments.

In this paper, we consider the principles of the MOLGENIS software suite [2], [3] in a new perspective, where it can be used as a separated module to generate all the necessary instruments for tracing data and computations and collecting logging information from different computational environments. We allow the users to decide how they want to orchestrate these instruments in different middleware systems, although we supply a default management solution, which includes database and a "pilot-job" framework to run analysis jobs. In addition, we show how the system can be applied to run NGS and imputation workflows in the cluster and grid environments.

This paper is structured as follows. Section II reviews related work in the context of conventions used in MOLGENIS Compute and attempts to deploy the workflow management systems to different computational back-ends. Section III describes the workflow design model used for workflow generation and the generation process. Section IV presents the workflow deployment and the design of the "pilot-job" framework used for workflow execution. Section V explains implementation details of creating analysis protocols, which are suitable for execution in different back-ends. Section VI provides details on NGS and imputation workflows. Section VII discusses our practical experiences, benefits and drawbacks of using the system. Section VIII offers our conclusions.

II. Related Work

We divide the related work into two sub-domains that describe work in (1) workflow design and generation and (2) workflow execution and deployment.

A. Workflow generation and design

Software generators are systems that generate other software and software specifications described in a model language play a role of the input for the generating. Our workflow model is described in Section III and used as an input for the workflow generation. As an output, we expect a kind of generated workflow management system, which is able to run workflows in a distributed computational environment.

We have gained a lot of experience in generating software infrastructures to manage and process large bioinformatics datasets [4], [5], [6]. The MOLGENIS toolkit [7] provides bioinformaticians with a simple language to model biological data structures and user interfaces. The MOLGENIS generator suite automatically translates these models into a feature-rich, ready-to-use web application including database, user interfaces, exchange formats, and scriptable interfaces. This "model-driven" method ensures re-use of best practices and improves quality because the modelling language and generators are shared between all MOLGENIS applications, so that errors are found quickly and improvements are shared easily by a re-generation. A plug-in mechanism ensures that both the generator suite and the generated product can be customized just as much as hand-written software.

In our previous work in the workflow management, we aimed to combine computational and data management in a single system - MOLGENIS Compute [8]. Then, we extended the initial solution with more specific for NGS analysis meta model, which allowed us to specify data provenance and workflow execution logic efficiently [9]. In our NGS-specific solution, we consider the computational cluster as a back-end, where all the computation take place. The users communicate with the system through the generated web-interface. Some logging information is usually hidden in a user interface, which makes a view on the data more compact and easier to comprehend. However, in some cases, such as debugging a new workflow or deploying it in a new environment, users may need this hidden logging information from an operational system or analysis tools. Expert users would like to have a direct and easy access to all the log files produced and it should not be obliged to do this through the web-interface, which requires extra effort to implement. This time we want to provide both command-line and web interface to the workflow management.

B. Workflow execution and deployment

If, in the previous work, we focused on generating compute management software for PBS clusters [10], here, we want to run workflows in different back-end environments. There are a number of projects such as MOTEUR [11] and TavernaPBS [12], which try to deploy the Taverna workflow management system (WMS) [13] to the grid and cluster environment respectively. Furthermore, adding the BioMoby [14] plug-in into Taverna can make running biological workflows and data provenance even more structural. Another workflow system Galaxy [15], which is specialised in running bioinformatics workflows can be configured to run analysis in Portable Batch System (PBS) or Sun Grid Engine (SGE) clusters, but we are not aware of any projects, which connect Galaxy to the grid middleware.

Besides WMS, bioinformatics analyses can be executed in distributed computational environments using web-based portals, such as P-GRADE Portal [16], which supports development and submission of distributed applications executed on the computational resources of various distributed computing infrastructures (DCIs) including clusters, service grids via web browser. The portal architecture consists of several layers including the presentation layer with a web interface, middle layer with different services and architecture layer with available cluster/grid middleware connectors. In our approach, we also want to communicate with the system through the generated web presentation layer, but we minimise another middleware layer between cluster/grid services and our system and to keep a minimal implementation to communicate to back-ends middleware.

For the workflow deployment, instead of the service based strategies as proposed in Taverna, where actual analysis tools are wrapped into web-services or other "black-box" components, we would like to use tools directly in both cluster and grid settings. This will allow us to use analysis tools without loosing any functionality, that can be hidden in a wrapper interface. We want to automate tool installation process which can be suitable for all, if it is possible, environments, that we include into our back-end compute infrastructure.

To summarize, we (1) re-use best practices from the Molgenis database generator to generate experiment data and workflow models for users and (2) fulfil a direct connection to cluster/grid infrastructures for bioinformatics.

III. Workflow design and generation

A. Workflow design

In this new effort, we would like to model bioinformatics workflows to run them in a unified way in different computational environments. We can divide the modelling task into two sub-tasks, such as

- workflow modelling, which match describing workflow to run in different computational back-ends
- data modelling, which covers different bioinformatics analyses (*e.g.* NGS or imputations)

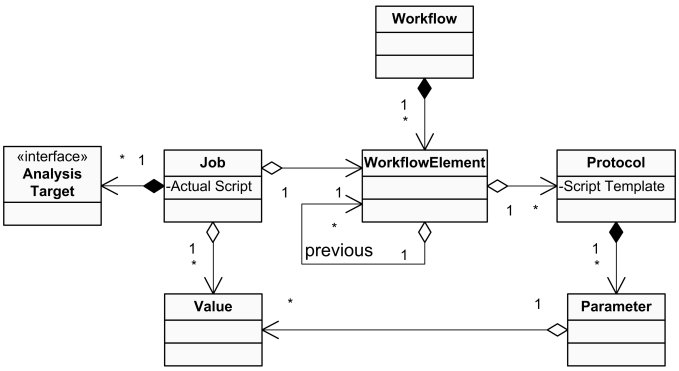

Fig. 1: Core of the computing model

1) Workflow modelling: The core of our workflow design remains stable [8], [9]. Our main goal is to keep the model

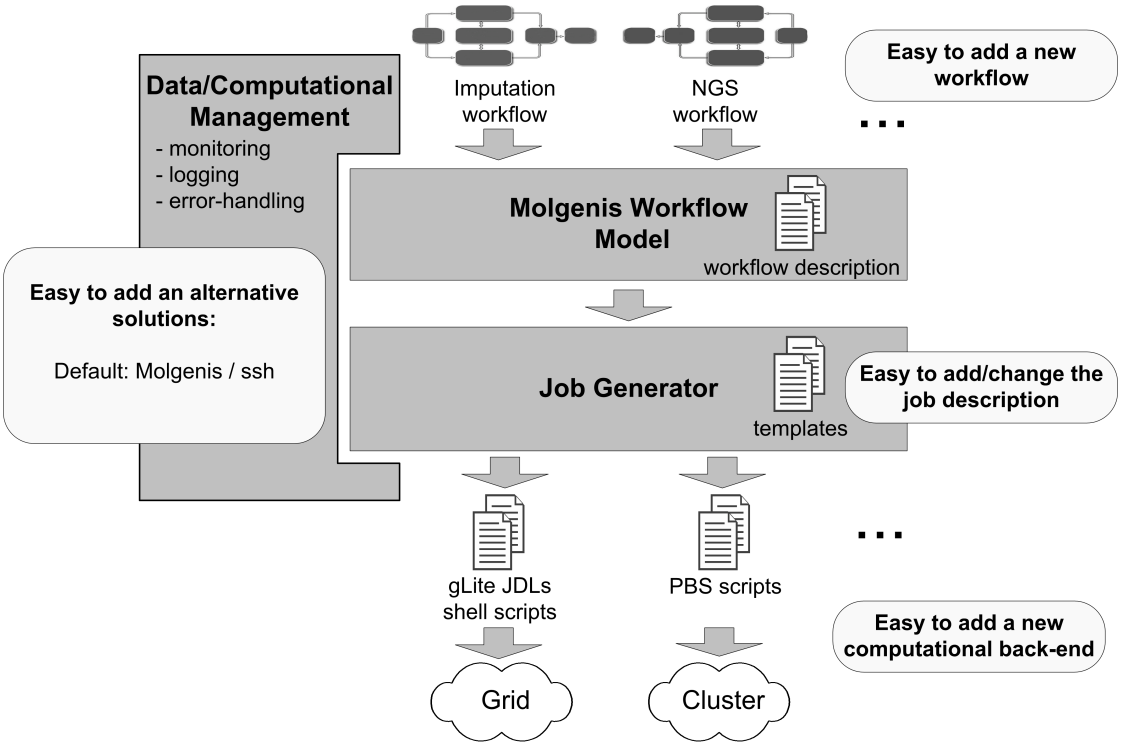

Fig. 2: Generation procedure using Molgenis job generator (a command-line version)

as simple as possible and avoid abstract entities. The model is shown in Figure 1.

The *Workflow* element includes all individual analysis *WorkflowElements* and represents the succession of the analysis. The core of the model are analysis script templates stored in the *Protocols*. A *Protocol* has its *Parameters*. *Parameters* can be of different types, such as workflow parameters, environment parameters and user parameters. During generation, the template parameters will be filled with actual *Values*. Then, the resulting script will be combined with supplementary headers and footers, which are specific for different back-ends, and stored as an analysis *Job*. The generation using templates is discussed next.

2) Data modelling: Our NGS model is described in details in [9]. It is based on the laboratory process when using an Illumina HiSeq 2000 machine, which produces the input data for our analysis workflows. It consists of such elements as *Projects*, *Samples*, *DNA Libraries*, *Flowcells*, *Barcodes etc.*

We introduce one common *Analysis Target* interface to enable us later to refer uniformly to subjects from different analyses. *Analysis Targets* can have complex relations between themselves. For instance, in our NGS workflow, some analyses are executed per *Barcode* being *Analysis Target*, but others are executed per *Project* or *Sample* being *Analysis Target*, which includes all *Barcodes*. Hence, it is very important to specify *Analysis Targets* and their relations for every analysis element in the workflow.

B. Analysis jobs generation

In general, an analysis, which is intended for execution in the cluster or grid, consists of many chunks and these chunks are arranged into workflows. The workflow structure is essential to run analysis parts in the correct order. However, usually, all the workflow parts can be generated instantly for the later execution. An exception is a case than a number of outputs (*i.e.* output files) is not know beforehand and these files should be used as inputs in further steps of the workflow.

We support two implementations for the workflow generation: (1) a database version and (2) a command-line version. which uses files instead of database. In the database version, a workflow is described in the MySQL database, which is generated from the model (Sec. III-A1). In the command-line version, a workflow is described in files:

- *workflow.csv*: a file with a sequence of workflow elements,
- *paratemers.csv*: a file with workflow parameters, and
- *protocols*: a folder with templates of workflow protocols.

These files match the database model and can be imported/exported into the database system. *Analysis Targets* also can be either selected as database records through the generated web interface or listed in the input *parameters* file. The workflow description and *Analysis Targets* are an actual input for generating analysis jobs.

In the database version jobs are generated as records in the database. The generation scheme of the command-line version is presented in Figure 2. Here, jobs are generated as script files in the specified directory. These script files are ready for submission to a specified in the generation back-end. For this, we provide supplementary headers and footers for different computational back-ends. These headers/footers are written as

templates and stored as files. They are filled with parameters during generation. Headers/footers are used to specify *e.g.* job wall-time, memory requirements, job logging. All that is needed to generate jobs for a new back-end, is to add new headers/footers templates for it. The generator source code stays the same. In addition, we provide a possibility for users to submit single analysis jobs or jobs with dependencies to a specified back-end.

In the database version, we are using "pilot-jobs" to run and monitor analysis jobs. This solution requires usage of the web-server and DB, that is discussed next.

IV. Workflow deployment and execution in different computational environments

A. Workflow deployment

We define a workflow deployment method that allows us easily add a new computational back-end and run workflows there. Here, by workflow deployment, we mean possibility to execute all workflow analysis steps in all available computational resources, such as clusters and grids, that does not require changes in analysis structure or protocols. This is very important to ensure reproducibility of workflow results, even if the analysis is executed in a new environment. Previously, we have executed workflows only in the computational clusters. Hence, it was sufficient to set up the same execution settings in all clusters to ensure reproducibility. In this new effort, we reuse the environment modules package [17], that provides a dynamic modification of a user's environment via module files.

Two actions should be performed to install a new software module into the system: (1) a software should be deployed on an execution site and (2) a module file, which describes the software should be added to the system. For a module deployment, we submit deploy scripts as simple cluster/grid jobs to a cluster/grid scheduler. An example of the deploy job, that deploys GATK software for the `glite-WMS` scheduler can be found at the Dutch project for biobank repository [18]. We define two types of deployment scripts: (A) pre-built and (B) on-site build. The (A) pre-build deployment has the following logical steps:

1) (manually) create a module file, that contains all changes to environment variables required to make the software working
2) (manually) build the software binary files
3) (manually) compress the built software and module file and upload them to an online repository
4) (cluster/grid job) downloads packaged software from the repository and decompress it
5) (cluster/grid job) moves the software to the right directory and ensures the permissions are right
6) (cluster/grid job) moves the module file to the right directory
7) (cluster/grid job) cleans-up, checks an environment variables and sends the deployment report

The (B) on-site build software deployment has a few differences. A cluster/grid job downloads the source code files of the software from the online repository and build the software binary on the cluster/grid site. This approach is needed when software, which should be installed, has dependencies on the system libraries, *e.g.* the R software for statistical computing. When deployment is completed, the deployed modules can be initialized using the following statement in analysis scripts:

```
module load your_software_module/version
```

The loaded in the module software should be invoked directly without specifying its classpath. Hence, we ensure that the same software is used for all analyses and the analysis scripts for different back-ends have exactly the same listings to access tools used in the analysis. As a result, the generated analysis scripts (Sec. III-B) will be executed in all computational sites in the same way and produce identical results.

B. Workflow execution

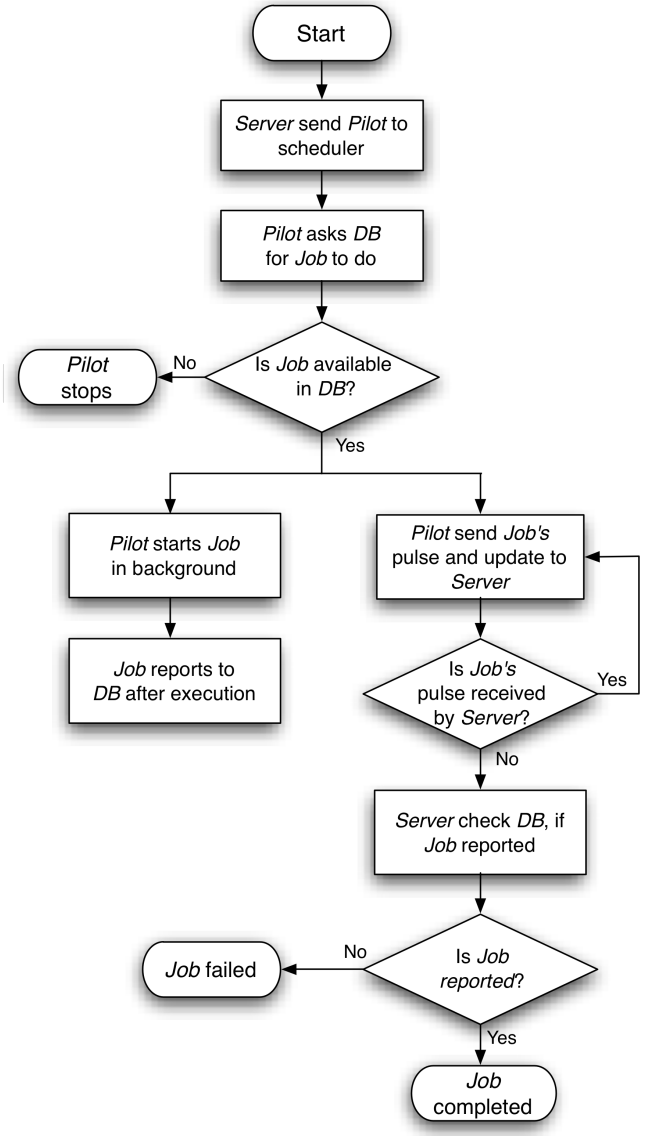

Fig. 3: Job execution using the MOLGENIS "pilot job"

We aim to run workflows in different computational back-ends in a unified way. Different back-ends can have different scheduling middleware and we like to minimize efforts to switch to new ones. Furthermore, we learnt that distributed

scheduling systems cannot be always fully reliable in practice. Hence, we want to minimize our dependency on a back-end middleware when running large bioinformatics analyses.

One of possible solutions is to use a ”pilot-job” framework. Our proposed ”pilot-job” framework is less exhaustive than proposed by Luckow *et al.* [19] and it does not cover all components of the complete ”pilot-job” frameworks, such as ”pilot-data” or ”pilot-resource”.

In our scenario, the ”pilot-job” (further *Pilot*) is sent to a computational back-end from the MOLGENIS web-server, which also contains the database with actual generated analysis jobs (Sec. III-B). A back-end scheduler put *Pilot* into a queue for execution. When *Pilot* is started in the execution node, it calls back to the MOLGENIS web server using cURL and asks for an available analysis job to execute. If any job is available, it is given to *Pilot*. *Pilot* starts the actual analysis by starting analysis script in the back-ground and continues in the main thread to send notifications and updates about the analysis job status back to the web server. When analysis is finished, its log files are send back from the analysis script to the database also using cURL. The analysis job is counted as completed if both the *Pilot*, as a monitor, and the job itself reported a successful completion. Otherwise, the job is counted as failed. The whole execution process using the proposed ”pilot-job” scenario is shown in Figure 3.

This approach works for both the glite-WMS grid and PBS/SGE cluster middleware. The only differences are the ”pilot-job” itself and a command line, which is used to submit the ”pilot-job” to the scheduler. In the current implementation, ”pilot-jobs” are submitted via the web-generated for every `user@backend` pilots dashboard (Fig. 4).

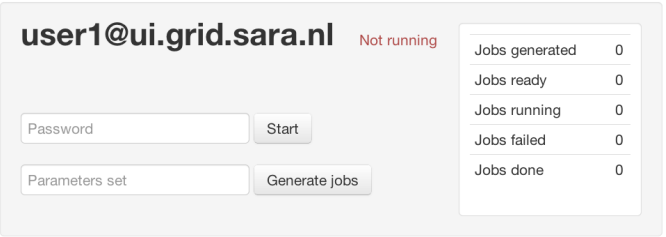

Fig. 4: Example of the web MOLGENIS pilots dashboard.

The MOLGENIS database contains a number of back-end records, which include such information as the back-end address, the back-end type (*e.g. gLite grid, PBS cluster*), the username and the path to the file, which will be used for the pilot submission (*e.g. .jdl, .sh files*). The dashboard user can generate jobs for submission to the specific back-end and, then, submit and monitor their execution after entering his password. In the future, we are planning to try creating separated dashboards for every analysis run. The resubmission option becomes visible in the dashboard, if failed jobs exist.

The actual ”pilot-jobs” are simple shell scripts. They are defined per beck-end middleware and embody allocation of the back-end resources (*e.g.* wall-time, memory). The ”pilot-jobs” source code is available at the MOLGENIS repository [20]. The dashboard user can choose specific ”pilot-jobs” for submission of long, short, CPU or memory demanding jobs. To summarize, we propose the solution, which trivially enables using a new middleware in our system and does not requires any deep study of a new middleware technical details.

V. Implementation detail on analysis protocol templates

As discussed above, actual scripts are generated from templates stored in *Protocol* elements or protocol files, where template parameters are filled in with actual values (Sec. III-A1). Here, we present the listing of the protocol template of the *BwaAlign* operation, as a protocol example.

```
//header
#MOLGENIS walltime=15:00:00 \
nodes=1 cores=4 mem=6
#FOREACH leftbarcodefqgz

//tool management
module load bwa/${bwaVersion}

//data management
getFile ${indexfile}
getFile ${leftbarcodefqgz}

//template of the actual analysis
bwa aln \
${indexfile} \
${leftbarcodefqgz} \
-t ${bwaaligncores} \
-f ${leftbwaout} \

//data management
putFile ${leftbwaout}
```

The template consists of several parts. Lets us look at them in detail. The script `header` is used only in the command-line version of MOLGENIS Compute (Sec. III-B) for workflow generation from files. In the command-line version, the final analysis commands are enveloped with the header and footer of supplementary commands for a specific back-end. The `#MOLGENIS` header is used for generation of these back-end specific headers/footers, which contain job specific parameters used for job scheduling, such as a job wall-time, number of required memory, cores to run the analysis *etc*. The `#FOREACH` specifies the *Analysis Target* used in this protocol (Sec. III-A1).

Next, the `tool management` section is listed, which is discussed in Section IV-A. The `bwa` module is loaded in this example. Further, there are the `data management` and `actual analysis` sections. The `data management` ensures transfer of required for analysis files to the location (computational node) in the grid or cluster, where actual computations take place. Functions `getFile` and `putFile` are differ per a back-end type. For example, for the grid back-end, these functions implement file transfer using `srm-commands` from and to the central storage used in the National Computing Grid for Life Sciences [21], where we run our computations. This transfer is needed to keep analysis results saved, after analysis is finished. The listings of data transfer files are also available at the MOLGENIS repository. [20].

The `actual analysis` section contains the template of an actual analysis command, which is filed with values

during generation (Sec. III-B). Variables in curly brackets are *Parameters* in our model. Some *Parameters* can consist of a combination of others and, in turn, are constructed using templates. For example, *Parameter* `leftbwaout` has the following value:

```
${leftfile}.bwa_align.${indexfileID}.sai
```

So, a variable can be a complex combination of other variables. The generated command the *BwaAlign* example is listed below:

```
bwa aln \
human_g1k_v37.fa \
121128_SN163_0484_AC1D3HACXX_L8_CAACCT_1.fq.gz \
-t 4 \
-f 121128_SN163_0484_AC1D3HACXX_L8_CAACCT_1.bwa_align.human
_g1k_v37.sai
```

Hiding supplementary operations for tool and data management, which are specific per back-end, outside the generated scripts, allows us to run the same scripts in different back-end. Using the module system to access analysis software and some additional data needed for analysis, such as large reference sets and genome builds, guarantees the identical analysis results, when running the analysis in different back-ends.

VI. Examples of available workflows for the gLite-WMS grid and clusters

We created a library of several public available workflows [22]. Two of them are fully tested in the National Computing Grid for Life Sciences, which uses gLite-WMS as a scheduling middleware, PBS/SGE/BSUB clusters and on a local machine.

A. NGS alignment

The first workflow is the NGS alignment and SNP calling workflow, which is comprised of best-practice open-source software packages used in multiple institutions leading to 23 analysis steps. It consists of four major parts:

(1) Alignment: The alignment is performed using Burrows-Wheeler Aligner BWA. The produced SAM file is converted to a binary format using Picard and sorted afterwards.

(2) Realignment: This part of the workflow duplicates reads are marked using Picard. Afterwards, realignment is performed around known insertions and deletions using the Genome Analysis ToolKit GATK. If reads are re-aligned, the fix-mates step will update the coordinates of the reads mate.

(3) Quality score recalibration: The base quality scores of a read a re-calibrated using covariate information are performed here. This method takes several covariates like cycle, dinucleotides, readgroup *etc.* into account and recalculates the quality scores, leading to reads being re-written with better empirical quality scores.

(4) Variant calling: The last part of the workflow performs indel and SNP calling using the GATK. The output of the pipeline are two VCF files, one with indels and one containing SNPs, ready for downstream analysis.

Expected time to run this workflow per *Sample* is about two days, depending on whether we are doing the exome or whole genome sequencing.

B. Imputation

The second workflow is imputation of the genome wide association study (GWAS). This workflow consists of four steps:

(1) Preparing the reference: Here, the reference is created from the Variant Call Format (VCF) of the genome reference (This steps is executed per the reference set).

(2) Preparing and checking the quality of the study data: In this step, all alleles are aligned to the reference genome and performs quality control.

(3) Phasing: Here, the data is phased using SHAPEIT2 [23], this is done per chromosome. The phasing also can be done once for a specific study.

(4) Imputation: This step consists of imputing the phased data with IMPUTE2 [24] and concatenation of the results per chromosome.

Normally, we split the dataset per 5 megabases regions and per 500 samples for one imputation job. The number of imputation jobs can be calculated using Formula 1. This step is executed per study.

$$Chunks = \sum_{chr=1}^{22} \frac{Lenght_{chr}}{5Megabase} * \frac{Samples}{500} \qquad (1)$$

For genome build 37 and 500 samples this yields 589 chunks. The execution time mostly depends on the used reference set. We observed an overage of 50 minutes to impute one chunk, when running the workflow in the National Computing Grid for Life Sciences [21]

VII. Discussion

A. Scaling bio-analysis

Figure 5 summarizes our view on the bio-analysis scaling process.

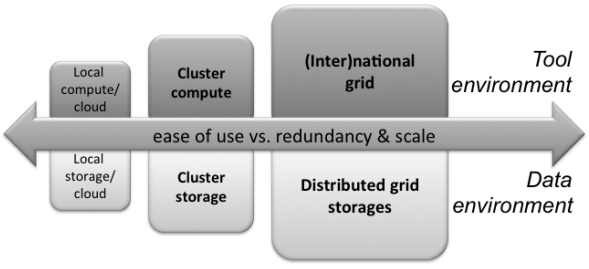

Fig. 5: Scaling computational environment

Scaling can be split up into two environment spaces: (1) analysis tools and (2) analysis data. Both tool and data should be available in computational back-ends, where we want to run analyses. The module deployment approach (Sec. IV-A) solves the software availability problem in both cluster and grid environment. Deployment scripts should be run in all sites of the grid for the grid deployment. The data availability in the grid can be solved by uploading input data and intermedia analysis results to the central grid `srm` storage, which is

available to all grid sites. We have found a good practice to define a `root` directory both in grid and clusters used and keep the directory structures starting from the `root` identical in all back-ends. Also, automating of the file transfer between the `srm` grid storage and local clusters outside the grid can be very useful.

B. Pilot implementation

We tried to keep modelling and implementation of our "pilot-job" approach straight forward, therefore, robust. We found that sending the "pulse" or the status of the actual analysis is an important functionality of the "pilot-job". It is very useful, because we intend to run long (2-10 hours) analysis jobs and we want to know about possible analysis failure as soon as possible. We can improve usage of resources allocated for a "pilot-job" by sending several analysis jobs to one "pilot-job", if the analysis jobs fit the "pilot-job" wall-time. We meet conditions of "pilot-data" by the proper workflow design, where we divide analysis data on chunks if it is needed. However, it can be interesting to try out other third party "pilot" solutions.

C. Ensuring correctness when running NGS and imputations in the distributed environment

Running analysis in the grid is even less reliable than in the cluster, since it includes extra file transfers and less control of remote execution nodes. Furthermore, we do not want to rely only on the context of the log files produced by the analysis tools for NGS and imputations with large datasets to analyse.

For NGS analysis, we kept a number of DNA sequence reads constant from input workflow files to the final results of the analysis. We did not remove any reads during filtering, instead we just marked them. So, the number of reads stayed constant during the whole analysis and checking this postcondition after analysis guarantees its correct completion. The number of reads in sequence data files can be calculated using the Genome Analysis Toolkit GATK [25]. We performed the reads calculation only in the end of the pipeline due to computational overhead. For imputation with minimac, a number of samples in input and output files is a good indication of a successful analysis completion. A number of samples is equal to a number of lines in these files. It should be equal in input and output files and can be easily checked.

D. Workflow modelling and visualization

Having the database and web user interface generated from the model, it is not surprising that we chose a simple table to show the workflow structure. An example workflow visualization in MOLGENIS compute is shown in Figure 6. Every line of the table shows *WorkflowElement*, its *Protocol* and previous *WorkflowElements*, if they exist. Generated *Jobs*, their *Parameters* and *Values* are presented in separated tables in the same style.

Many other established WMS visualize workflows as graphs, where the nodes are analysis steps and the edges are interactions and constraints between analysis steps. Still, proposed visualization do not cover all analysis aspects needed to optimize workflow execution. As one of our next development direction, we want to apply some visual analytics

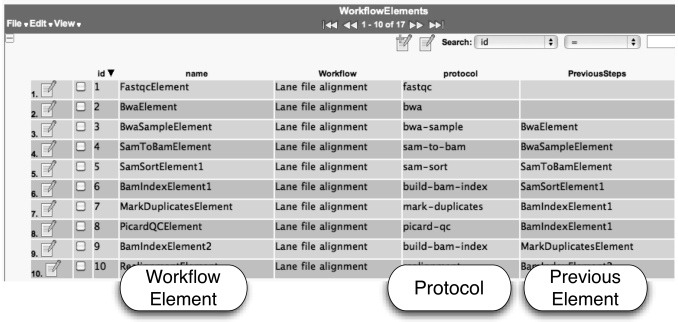

Fig. 6: Showing workflow structure in the MOLGENIS framework [8]

methods used in software field, such as combined UML (unified modeling language) diagrams and "quality" metrics, to enhance understanding and sharing of workflows, and ease workflow analysis and design [26].

VIII. Conclusion

We proposed a lightweight collaborative analysis environment for bioinformatics service teams, where the analysis to be computed in different back-end environments, such as computational clusters and grids, is specified in a unified way. For this, we use one model, which is suitable for different workflows, such as the NGS alignment and SNP imputation. All analysis jobs can be generated in the database and executed using a "pilot-job" framework, or they can be generated as shell scripts with back-end specific headers and footers.

We would next like to create workflow importer/exporter to interchange workflows between different workflow management tools, such as Galaxy or Taverna. Hence, we can re-use more externally defined workflows and easily share workflow defined in our group. A second essential direction is to study the practical effectiveness of running workflows in different computational back-ends using described in [26] visual analytics methods and learning from the best practices and experiences.

Acknowledgment

The authors would like to thank the BBMRI-NL Rainbow Project 2 (http://www.bbmri.nl), the Target project (http://www.rug.nl/target) and the BigGrid eBioGrid project (http://www.ebiogrid.nl). Also, the authors thank Erwin Winder for implementing the pilots dashboard generator.

References

[1] M. DePristo and M. Daly, "A framework for variation discovery and genotyping using next-generation dna sequencing data," *Nature Genetics*, vol. 43(5), pp. 491–498, 2011.

[2] M. Swertz and R. Jansen, "Beyond standardization: dynamic software infrastructures for systems biology," *Nature Reviews Genetics*, vol. 8:3, pp. 235–43, 2007.

[3] ——, "The molgenis toolkit: rapid prototyping of biosoftware at the push of a button." *BMC Bioinformatics*, vol. 11:12, 2010.

[4] ——, "Xgap: a uniform and extensible data model and software platform for genotype and phenotype experiments," *Genome Biology*, vol. 11:27, 2010.

[5] Y. Li and R. Jansen, "Global genetic robustness of the alternative splicing machinery in caenorhabditis elegans," *Genetics*, vol. 186(1), pp. 405–10, 2010.

[6] Y. Li and M. Swertz, "Designgg: an r-package and web tool for the optimal design of genetical genomics," *BMC Bioinformatics*, vol. 10:188, 2009.

[7] Genomics Coordination Center, Groningen, "Molgenis web-site," 2011, http://www.molgenis.org.

[8] H. Byelas and M. Swertz, "Towards a molgenis based computational framework," *in proceedings of the 19th EUROMICRO International Conference on Parallel, Distributed and Network-Based Computing*, pp. 331–339, 2011.

[9] ——, "Introducing data provenance and error handling for ngs workflows within the molgenis computational framework," *in proceedings of the BIOSTEC BIOINFORMATICS-2012 conference*, pp. 42–50, 2012.

[10] Adaptive Computing, "Torque resource manager," 2012, http://www.adaptivecomputing.com/products/open-source/torque/.

[11] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, "Flexible and efficient workflow deployment of data-intensive applications on grids with moteur," *Int. J. High Perform. Comput. Appl.*, vol. 22, no. 3, pp. 347–360, 2008.

[12] University of Virginia, Center for Public Health Genomics, "Tavernapbs," 2012, http://cphg.virginia.edu/mackey/projects/sequencing-pipelines/tavernapbs/.

[13] T. Oinn and M. Greenwood, "Taverna: lessons in creating a workflow environment for the life sciences," *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE*, vol. 18:10, pp. 1067 – 1100, 2005.

[14] E. Kawas and M. Wilkinson, "Biomoby extensions to the taverna workflow management and enactment software," *BMC Bioinformatics*, vol. 7:523, 2006.

[15] D. Blankenberg and J. Taylor, "A framework for collaborative analysis of encode data: making large-scale analyses biologist-friendly," *Genome Res.*, vol. 17:6, pp. 960 – 4, 2007.

[16] P. Kacsuk, "P-grade portal family for grid infrastructures," *Concurrency and Computation: Practice and Experience journal*, vol. 23:3, pp. 235–245, 2012.

[17] P. O. J.L. Furlani, "Abstract yourself with modules," *Proceedings of the Tenth Large Installation Systems Administration Conference (LISA '96)*, pp. 193–204, 1996.

[18] BBMRI.NL, "the dutch project for biobank," 2012, http://www.bbmriwiki.nl/svn/ebiogrid/modules/GATK/1.0.5069/.

[19] A. Luckow, "Saga bigjob: An extensible and interoperable pilot-job abstraction for distributed applications and systems," *Proceedings of the Tenth 0th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010.

[20] MOLGENIS team, "the molgenis github repository, pilot sources," 2012, http://github.com/molgenis/molgenis_apps-legacy/tree/testing/modules/compute/pilots/.

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

[22] MOLGENIS team, "the molgenis workflow repository," 2013, http://github.com/molgenis/molgenis-pipelines/tree/master/compute4.

[23] O. Delaneau and J. Marchini, "Improved whole-chromosome phasing for disease and population genetic studies," *Nature Methods*, vol. 10, pp. 5–6, 2013.

[24] B. Howie and J. Marchini, "A flexible and accurate genotype imputation method for the next generation of genome-wide association studies," *PLoS Genet*, vol. 5, p. e1000529, 2009.

[25] The Genome Analysis Toolkit, "Broad institute," 2011, http://www.broadinstitute.org/.

[26] H. Byelas and M. Swertz, "Visualization of bioinformatics workflows for ease of understanding and design activities," *Proceedings of the BIOSTEC BIOINFORMATICS-2013 conference*, pp. 117–123, 2013.