Available on-line at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Scaling of Biological Data Workflows to Large HPC Systems – A Case Study in Marine Genomics –

Thomas Röblitz[a,*], Ole W. Saastad[a], Hans A. Eide[a], Katerina Michalickova[a], Alexander Johan Nederbragt[b], Bastiaan Star[b]

[a]Department for Research Computing, University Center for Information Technology (USIT), University of Oslo, P.O. Box 1059, Blindern, 0316 Oslo, Norway
[b]Center for Ecological and Evolutionary Synthesis, Department of Biosciences (CEES), University of Oslo, P.O. Box 1066, Blindern, 0316 Oslo, Norway

Abstract

Sequencing projects, like the Aqua Genome project, generate vast amounts of data which is processed through different workflows composed of several steps linked together. Currently, such workflows are often run manually on large servers. With the increasing amount of raw data that approach is no longer feasible. The successful implementation of the project's goals requires 2-3 orders of magnitude scaling of computing, while achieving high reliability on and supporting ease-of-use of super computing resources at the same time. We describe two example use cases, the implementation challenges and constraints, the actual application enabling and report our findings.

1. Introduction

Advances in sequencing technology have promoted research aimed to better understand the relationships between genetics and phenotypic traits in many organisms, including a variety of fish species. For instance, as part of the CEES-led Aqua Genome project (in collaboration with NOFIMA [8] and CIGENE [2]), the genomes of 1000 Atlantic cod individuals will be sequenced with an aim to investigate the genomic basis of phenotypic traits that are – amongst others – related to feed efficiency, growth, age at maturation, disease resistance and product quality and simultaneously assess the importance of these relationships in wild populations. This work contributes to the ecological and economical sustainable development of aquaculture and fishery industries, which – in light of an impending global food crisis – play a crucial role in providing food security. Successful implementation of the project's goals requires 2-3 orders of magnitude scaling of computing for various data analysis workflows, while achieving high reliability on and supporting ease-of-use of super computing resources at the same time.

In such sequencing projects like the Aqua Genome project, use cases are composed of several steps linked together in a workflow. Each step addresses a specific part of an analysis beginning with sequenced data. Currently, such workflows are often run *manually* on large servers. With the increasing amount of raw data that approach – manually executing a workflow instance on a single large server – is no longer feasible. To satisfy the vast demands of compute capacity, many servers – typically found in a cluster or HPC system – must be used. In addition, the execution of a single workflow must be fully automated, thereby allowing large parameter sweep studies.

In a collaboration between the Center for Ecological and Evolutionary Synthesis (CEES) at the Department of Biosciences (University of Oslo, Norway) and the Department for Research Computing at the University Center for Information Technology (University of Oslo, Norway), we are working on the scalability of the comprehensive processing of such workflows. As a first step, we implemented two example workflows to run on the PRACE Tier-0 machine Curie [3]. The enabling work consisted of installing software tools, writing job scripts for the individual steps and performing tests with different parameters and configurations. It turned out that implementing a single workflow is relatively straight forward, but managing the logistics for the simultaneous execution of multiple workflows – staging data in and out, managing of storage space, and adapting to the specific scheduler configuration of the machine – remain a challenge. Ideally, these resource management tasks need to be hidden from the scientists. In the second step, we therefore explored the use of portals to further simplify the access to large HPC machines.

*Corresponding author.
e-mail. thomas.roblitz@usit.uio.no

2. Use cases

We considered two example workflows which are described briefly below. The high-level description does not include any implementation specific details, yet it gives an overview on the analysis steps involved in the workflow. The use cases describe the required processes that prepare for the single nucleotide polymorhpisms (SNP) variant calling phase – an essential analysis phase in many genomic applications.

2.1. Alignment and verification

Figure 1 shows the steps and data flow of the first use case. The encircled numbers illustrate a possible execution sequence. A letter at the start of an arrow specifies the number of outputs for a single input/parameter of the step of which the arrow originates from. Note, the control flow is not included in the description, and left out in the implementation (for a brief discussion of control flow aspects see Section 5.5.).

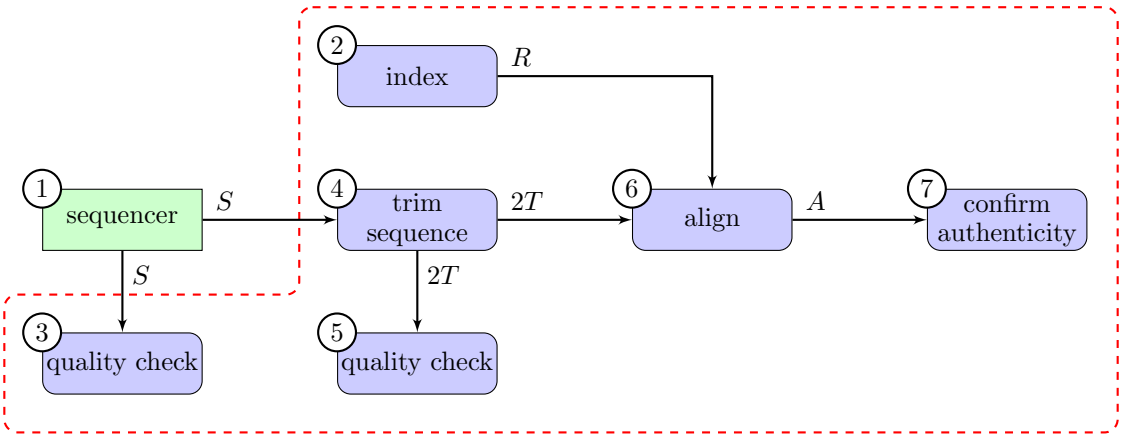

Fig. 1: Steps and data flow of the alignment and verification use case. The numbers show a typical execution sequence. Arrows illustrate data flow. The marked area is the subject of our case study.

We briefly describe the main characteristics of the use case below:

- The primary input to each workflow instance is a single sequence file generated by a sequencer (step ①).
- Step ② needs to be executed once for each reference genome. Let R be the number of reference genomes.
- For each sequence generated by the sequencer (step ①), the workflow can be executed several times applying different parameter sets. Let S be the number of sequences to be processed.
- Step ④ trims the primary input sequence from both ends using an adaptor sequence. Hence, a forward and a backward trimmed input sequence is generated, thereby doubling the number of input data files for follow-up steps.
- The parameters of a workflow instance are the adaptor sequences for trimming (T, step ④) and the alignment arguments (A, step ⑥).
- The overall number of workflow instances (W) to run is calculated by

$$W = R \cdot S \cdot 2T \cdot A\,.$$

  Hence, using a single reference genome ($R = 1$), 1000 sequences ($S = 1000$), a single set of adaptor sequences for trimming ($T = 1$) and a single set of alignment arguments ($A = 1$), a total of 2000 instances of the workflow are to be run.

  Each instance creates two alignments (one for the forward trimmed input sequence, one for the backward trimmed, step ⑥), runs three quality checks (1×step ③+2×step ⑤), and performs two authenticity confirmations (step ⑦).
- The overall outcome of running W instances are the called variants[1] between the reference genome R and the set of sequences S.
- Each step may require anywhere from one to many software packages. The software package(s) required by a step may be dependent or independent of the software being used by the other steps. For example, a quality check may use software packages independent of the one(s) used by any other step. On the other hand, the package to create an index (step ②) may be specific to the one used for the alignment (step ⑥).

[1] Called variants are the differences between the sequenced sequence and the reference genome.

2.2. Alignment and indels

Figure 2 shows the steps and data flow of the second use case[2]. The encircled numbers illustrate a possible execution sequence. A letter at the start of an arrow specifies the number of outputs for a single input/parameter of the step of which the arrow originates from. Note, the control flow is not included in the description, and left out in the implementation (for a brief discussion of control flow aspects see Section 5.5.).

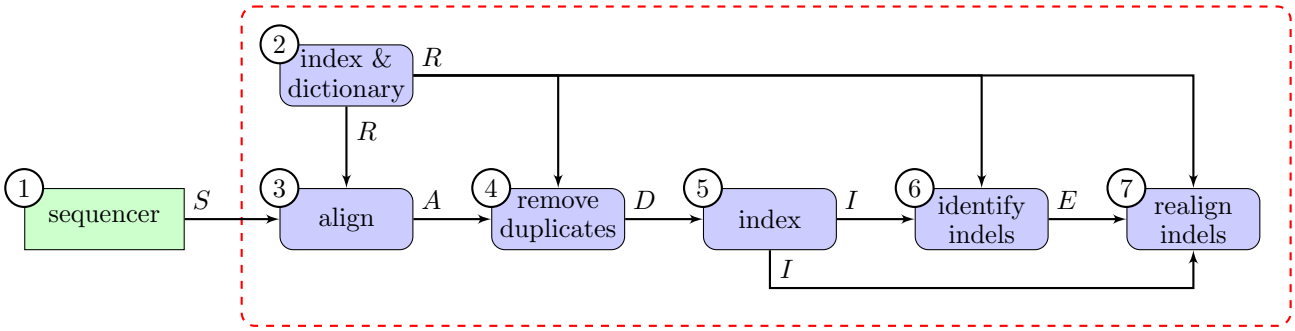

Fig. 2: Steps and data flow of the alignment and indels use case. The numbers show a typical execution sequence. Arrows illustrate data flow. The marked area is the subject of our case study. Note, for the sake of simplicity quality checks (cf. Fig. 1) are not shown and left out in the implementation.

We briefly describe the main characteristics of the use case below:

- The primary input to each workflow instance is a single sequence file generated by a sequencer (step ①).
- Step ② needs to be executed once for each reference genome. Let R be the number of reference genomes.
- For each sequence generated by the sequencer (step ①), the workflow can be executed several times applying different parameter sets. Let S be the number of sequences to be processed.
- The parameters of a workflow instance are the alignment arguments (A, step ③), the remove duplicates arguments (D, step ④), the identify indels arguments (I, step ⑥), and the realign indels arguments (E, step ⑦).
- The overall number of workflow instances (W) to run is calculated by

$$W = R \cdot S \cdot A \cdot D \cdot I \cdot E\,.$$

  Hence, using a single reference genome ($R = 1$), 100 sequences ($S = 100$), a single set of alignment arguments ($A = 1$), one argument for removing duplicates ($D = 1$), one parameter set for identifying indels ($I = 1$), and a single argument for realigning indels ($E = 1$), a total of 100 instances of the workflow are to be run.
- The overall outcome of running W instances are the called variants between the reference genome R and the set of sequences S.
- Each step may require anywhere from one to many software packages. The software package(s) required by a step may be dependent or independent of the software being used by the other steps.

2.3. Implementation challenges

Clearly, the two use cases are very similar in most aspects. Therefore, we do not distinguish between them in the following. The implementation challenges are:

C1 to use the software as is, i.e., to not employ any modifications to code (to improve performance, to lower resource consumption, or to make it work at all, etc.);

C2 to exploit parallelism at different levels of granularity: individual steps, within a workflow (intra-workflow) and among several workflows (inter-workflow). Step parallelism can be achieved by using software packages which exploit threading (shared memory) or message passing (distributed memory), cf. Table 1 for an overview of the used software packages. Intra-workflow parallelism makes use of the partial order of the steps in a workflow, that is, by executing steps in a set of non-dependent steps in parallel. Inter-workflow parallelism exploits the fact that instances of a workflow are independent of each other because each instance processes a distinct input data file; *and*

C3 to manage resources and resource limits of the HPC system: number of jobs (queued/running), storage quota (e.g., home, work, scratch file systems), maximum number of cores, minimum/maximum RAM, maximum wall clock time, efficient use of core cycles.

[2]Indels are insertions or deletions of bases in the DNA.

3. Implementation method & goals

An important constraint in our case study was to use existing bioinformatics software packages as they are, i.e., without any modification to their source code. First, since a variety of different packages can be used even in our relatively small use cases, we focused on a subset of tools. Second, we are neither experts nor developers of these tools. Therefore, we only installed tools if they were not yet installed and used the existing environment (Java, Python, Perl, etc.). Table 1 lists the software packages we used in our case study. Except for sysstat all packages were installed through our project.

Table 1: Overview of the used software packages. Most of these packages contain a large number of tools. The information about the package's capabilities for parallel execution (column Type) reflects the capabilities of the specific tools we used.

Name-Version	Description	Use case (Steps)	Type
bwa-0.7.5a	bwa is a package for mapping sequences against a reference genome.	1 (②, ⑥), 2 (②, ③)	threading
cutadapt-1.2.1	cutadapt is a tool for removing adapter sequences from DNA sequences.	1 (②)	sequential
FastQC-0.10.1	FastQC is a quality control tool for sequence data.	1 (③, ⑤, ⑦)	sequential
GATK-2.7.4	The Genome Analysis Toolkit allows variant discovery and genotyping.	2 (⑥, ⑦)	threading
mapdamage-2.0-33	mapDamage tracks and quantifies DNA damage patterns.	1 (⑦)	sequential
picard-tools-1.102	Provides tools for manipulating alignment data in the BAM format.	2 (②, ④)	sequential
SAMtools-0.1.19	Provides tools for manipulating alignment data in the SAM format.	2 (③, ⑤)	sequential
sysstat-9.0.4	Tools for collecting monitoring data about system performance.	1, 2 (all)	sequential

At the start of the case study three PRACE Tier-0 machines were available for implementing our use cases. We got accounts on Fermi (Cineca, Italy), JUQUEEN (Juelich, Germany) and Curie (CEA, France) and studied the documentation for these systems. On Fermi and JUQUEEN, the minimum core number of a test job is 1024 and 512, respectively, and for a production job it is 2048 and 8192, respectively. While there are software packages for bioinformatics that may utilize such core counts efficiently, most in our example use cases do not. Since we did not want to change the source code of the software packages, the only way to utilize that many cores in a single job would be to pack many different independent steps into a composite job (MPMD, multiple-program-multiple-data). That would not only have required a huge coordination effort (challenge C3, Section 2.3.), but also would have limited the flexibility in executing non-bulk runs. So we decided not to consider these two systems further. Fortunately, Curie has no such restriction on the core count and is quite similar with respect to hardware, software and job management environment to our local Tier-1 cluster Abel [1]. Hence, Curie was selected as the initial system in our case study.

Instead of implementing the use cases with a single script, or as a single job, we aimed at a framework with several additional features:

- quick development of new workflow steps and their integration into/addition to a use case,
- easy exchange of tools/software packages in a workflow,
- changing parameters without modifying scripts or config files,
- using templates for standardized handling of input and output data,
- easily and flexibly linking worklow steps to define execution dependencies,
- repeating steps with the same or changed parameters,
- supporting bulk operations to perform large parameter sweep studies,
- monitoring resource consumption of a step (CPU cycles, RAM),
- simplify porting to another system, *and*
- make it easy to use.

Table 2 lists tools that may be used as a basis to implement such a framework. Although, tools such as Unicore, Snakemake or Taverna support the majority of the required features we opted to use shell scripting and the Lifeportal [6] in a two phase implementation approach. In the first phase, we used shell scripting to incrementally implement individual steps and to link them together. We deliberately chose this technology because it provides a much higher level of control of the interaction with the underlying HPC system (particularly, the

connection to the resource management system), and yet it does not require additional middleware layers which add complexity and intransparency. Also, we believe it provides us much better insight into all the aspects that need to be considered in the scaling of workflows. Once we understood all those aspects better, we used the Lifeportal to implement the use cases to meet the requirement for an easy-to-use interface. The Lifeportal was selected because it is connected to our local Tier-1 cluster Abel to enable large scale runs.

Table 2: Tools for executing a single workflow instance and orchestrating multiple workflow instances.

Group	Examples	Description	Advantages	Disadvantages
Grid middleware	Unicore [14]	Toolkits to enable resource sharing across multiple sites.	backend is installed on Curie, supports workflows provides a GUI	deployment overhead, portability, level of control
Workflow engines	Taverna [13]	Provide means to describe and execute workflows on Grid, Cloud and cluster resources.	easy-to-use GUI, may use various remote resources	deployment overhead, portability, level of control
Portals	Lifeportal [6]	The Lifeportal is based on Galaxy [4] which is an easy-to-use portal framework for biomedical research.	tailored at bioinformatics, easy-to-use GUI, uses Tier-1 cluster Abel	no bulk operations, intransparent implementation of steps, level of control
Build tools	make, ant	Provide a means to describe data dependencies among the steps of a workflow.	available on many systems	proprietary description of a workflow, command line only, not interfaced with resource management system
Bioinformatics build tools	Snakemake [12], Ruffus [11]	Tools that implement a domain-specific language to mimic make but are tailored for bioinformatics use cases.	tailored at bioinformatics, may use clusters, lightweight	portability, command line only
Shell scripting	bash	Means to automate tasks at system level.	available on many systems, high level of control over various system aspects	proprietary description of a workflow, significant effort to implement logistics, command line only

4. Implementations

We implemented the two use cases on two systems – the PRACE Tier-0 supercomputer Curie (CEA, France) and the Lifeportal connected to the Tier-1 cluster Abel (University of Oslo, Norway).

Curie is a PRACE Tier-0 machine at CEA (France). It contains three different hardware partitions – super fat nodes, thin nodes and hybrid nodes. Hybrid nodes provide GPUs which were not used in our case study. Super fat nodes and thin nodes use different processors, numbers of processors and main memory per node. Table 3 shows the differences among the two hardware partitions that both were used in our case study.

Table 3: Hardware partitions on Curie and Abel

	Curie		Abel	
Feature	Super fat	Thin	Hugemem	Standard
CPU	Nehalem-EX 2.27 GHz	Sandy Bridge 2.7 GHz	Sandy Bridge 2.2 GHz	Sandy Bridge 2.6 GHz
CPUs per node	16	2	4	2
Cores per node	128	16	32	16
RAM per node	512 GB	64 GB	1024 GB	64 GB
Number of nodes	90	5040	8	652

Curie provides four file systems (cf. Table 4) each having a different purpose and exhibiting different characteristics. Note, neither WORK nor SCRATCH is backed up. For job management, Curie uses SLURM. Table 5 shows the main job classes and their limits.

Table 4: File systems on Curie and Abel

	Curie				Abel	
Feature	HOME	WORK	SCRATCH	STORE	/cluster	/work
Type	NFS	Lustre	Lustre	Lustre	FhGFS	FhGFS
Space limits	3 GB	1 TB	20 TB	–	200 GB	-
File limits	–	500 K	2 M	100 K	-	-
Bandwidth	low	10 GB/s	150 GB/s	90 GB/s	13 GB/s	15 GB/s
Used for	logistics	work	jobs in/out	backup	logistics, data	work, jobs in/out

Curie organizes the software (compilers, libraries and application codes) in modules. All basic software such as compilers were already installed. All application codes needed to be installed in a directory belonging to the user.

Table 5: Job management on Curie and Abel

	Curie			Abel	
Feature	long	normal	test	long	normal
Max CPUs	1024	–	–	2368	–
Max nodes	–	8	–	148	–
Max jobs submitted	10	300	2	–	–
Max jobs running	5	–	–	–	–
Max runtime	3 days	1 day	30 min	28 days	7 days

The **Lifeportal** is a Galaxy-based portal linked to our Tier-1 compute cluster Abel. Thus, all work done submitted through the Lifeportal interface is executed as a job on Abel. Moreover, all data that is uploaded to the Lifeportal and analyzed through Lifeportal jobs is stored on the file systems of Abel. For a user most of the system characteristics of Abel such as types of nodes, memory (cf. Table 3), file system quota (cf. Table 4), and job management (cf. Table 5) is hidden.

4.1. Implementation on Curie

First, we installed the needed software packages on Curie, and tested them with simple scripts and within interactive jobs. The first choice to be made was between implementing a whole workflow as a single job or as multiple jobs. For various reasons, we chose the latter approach: it enabled us to implement the workflow incrementally, it lowers the complexity of the implementation and debugging process, and it allowed us to tailor the resource and runtime requirements for each step individually. This is a crucial decision as it strongly affects the ability to meet the challenges intra-workflow parallelism (challenge C2, Section 2.3.) and efficient use of core cycles (challenge C3, Section 2.3.). Also, replacing steps, adding or removing them at a later stage will be easier with the chosen approach. For each step we prepared a job script and a configuration file that contains information for submitting and running a job. Two helper scripts `create.sh` and `submit.sh` were created to simplify the creation and submission of a workflow step, respectively. The former script creates the directory structure of a step (copies the configuration file, links input data). The latter script actually submits the job using the root of the directory structure created by `create.sh`. It also creates a new sub directory for each job run. Hence, by simply running the script `submit.sh` again, a new job is created. Structuring the jobs into individual directories makes it easier to manage the available storage quota (challenge C3, Section 2.3.). To adjust step and job parameters, the two scripts may be given command line arguments. A full workflow is generated by a third script that creates and submits all the individual steps. For dependent steps it uses a job parameter such that they are only started once their predecessors have finished.

One important aspect in submitting a job is always to specify meaningful resource and runtime requirements, i.e., they must not be to small/short to prevent interruption by the job management system and should only request little more than they actually need to ensure quick turn around times and to avoid wasting granted allocations. Observing the resource consumption of large scale runs manually is not a practical approach, because the input data and parameters may vary greatly in our use cases. Instead we needed a method that collects all relevant data automatically and analyzes it after a job has finished. The tool `pidstat` [9] collects all the information needed and can be run in user space. We reimplemented the script `create.sh` such that it combines two parts – the actual workload and the monitoring – into a single MPMD (multiple-program-multiple-data) job. The obtained monitoring data may be used to improve the resource consumption of jobs to meet many aspects of the challenge C3 (cf. Section 2.3.).

We then run thousands of jobs with different input sizes to test the implementation, and to determine resource and runtime needs of individual steps. For some parameters the alignment of the first use case required

very long runtimes. Therefore, we had to split the execution of the alignment into two jobs: one for the forward trimmed sequence and one for the backward trimmed sequence. Table 6 shows the results for these runs in separate rows. Hence, the decision to implement steps as individual jobs allowed us to manage the runtime limits on Curie (challenge C3, Section 2.3.).

A side result of our implementation was an improved version of the software package `mapDamage.pl`. A first version generated hundreds of thousands of files, rendering the software unusable given the file system constraints on Curie (cf. Table 4). A second version generated only a fixed number of files per run, but consumed huge amounts of main memory. Again the software was not usable. The developers quickly provided a fix that greatly reduced the needed memory and the software could be used. The responsiveness of the developers allowed us to avoid any modifications of the source code (challenge C1, Section 2.3.).

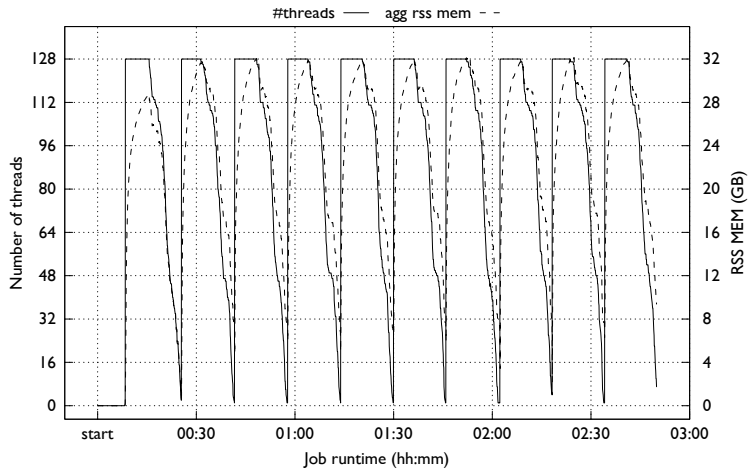

Fig. 3: Monitoring for a 127 core alignment job with the tool `bwa aln`.

We also run tests on large input files ($\approx$ 20 GB) where especially the alignment required very long runtimes even when using 127 cores on the super fat node partition (cf. Table 6). Figure 3 illustrates the memory consumption during the first three hours of an example run (total runtime was about 3 days). In total, all jobs consumed nearly 200 K hours on the super fat nodes, and about 10 K hours on the thin nodes. The aggregate amount of data being processed and created is approximately 2 TB.

During the test runs we improved our implementation by using symbolic links instead of copying input data, and used the file system STORE to free space on the file system WORK (challenge C3, Section 2.3.).

4.2. Implementation on the Lifeportal@Abel

We implemented the use cases on the Lifeportal by uploading sample input data and processed the data using the available tools. First, we started to use the production version of the Lifeportal. Figure 4 shows a screen shot of the portal with the main interface being used in our case study. Because some tools were not available on the production instance, we had to use the development instance where new tools are tested beforehand.

We installed missing tools and, for some steps, had to adapt XML templates and Python wrappers to create missing indices and dictionaries for GATK [5]. Note, our adaptations never involved changes in the source code of the actual tools. Thus, we met challenge C1 (cf. Section 2.3.). The adaptations are on a level comparable to the scripts developed for each step for the implementation on Curie (cf. Section 4.1.).

5. Findings and remarks for future directions

We present the findings we obtained during the implementations on both systems including an estimate of the achieved scalability, provide recommendations for reusing our results, and conclude with general remarks for future directions.

5.1. Findings for the implementation on Curie

We had no problems installing the needed software packages. Running GATK required a special key to prevent it from "phoning home" usage statistics. Since Curie nodes are on an internal network running GATK failed in the first attempt. Also `mapDamage.pl` [7] needed improvements by the developers to be usable. An advantage of Curie is the large quota of the file systems SCRATCH and STORE, and the easy interface to use the file system STORE as backup medium. User documentation on Curie is very good, detailed and contains many examples. Curie also has low minimum resource requirements for jobs and contains different machine sizes (super fat and thin nodes).

For our use cases the two file systems HOME and WORK were quite small even for development. Large scale production runs require significantly more space or much more sophisticated logistics for managing data.

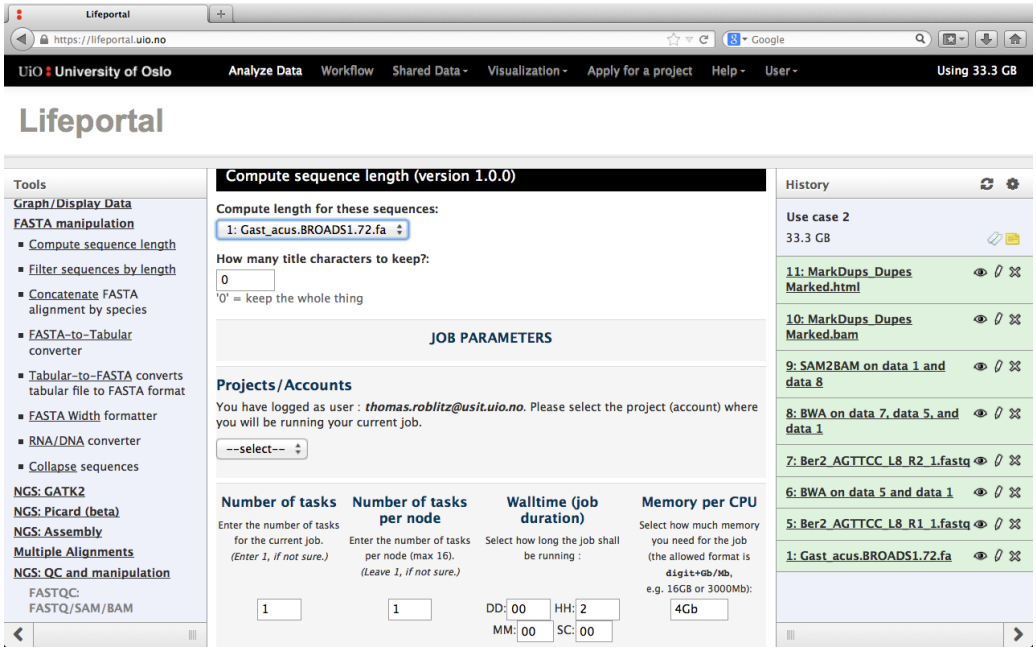

Fig. 4: Screen shot of the Lifeportal showing an example history for the second use case.

Managing data as well as remote access is only possible via password protected ssh/scp or gsissh/GridFTP. While the former should be easy to use for the use case owners, they will likely struggle with using Grid middleware. Also, the runtime limit of three days for the longest jobs was too small for some of our test jobs using real sized data sets and the maximum of 128 cores in a single super fat node (cf. Table 3). If neither of these limits can be lifted, alternative software tools that can make use of cores spread over multiple nodes are needed to increase the number of used cores and thereby reducing the runtime of a job.

5.2. Findings for the implementation on the Lifeportal@Abel

It was really easy (and fast) to start implementing the use cases on the Lifeportal. Also, repeating individual steps with the same or changed parameters cannot be made easier – a matter of a few mouse actions. Creating a workflow from a previous sequence of steps is very easy too.

By design the available tools may limit the choice or even hindering the implementation of a specific use case. So, a user may need support from a portal administrator to add needed tools. Debugging problems is difficult too. We also found that the available tools combine a number of programs, e.g., creating an index and performing an alignment. While that is convenient if it is done once, it may result in wasting compute hours and data storage.

5.3. Achieved scalability

We consider two aspects that limit the scalability of the simultaneous execution of a large number of workflow instances: the human coordination effort, and the footprint on system components.

Measuring the achieved scalability in connection with the coordination effort for humans is not trivial because, in the past, the workflow instances were – to a large extend – executed manually. Hence, no precise quantification of the coordination effort is available. However, it is fair to estimate that a single scientist can hardly manage more than 10 simultaneously running workflow instances. This claim is supported by the observation of runtimes of individual steps (cf. Table 6). Most steps finish after a short time and require attention by the scientist to initiate the next step. Because we completely removed the need for manual coordination of a workflows execution, many more instances can run simultaneously. For testing the functionality we easily spawned 60 workflow instances each containing five jobs executing a single step of the workflow. The number of workflow instances was only limited by resource constraints on Curie, namely the maximum number of submitted jobs (300 for the partition normal, cf. Table 5) and the available disk quota (1 TB for WORK, cf. Table 4).

The second aspect limiting scalability may be the footprint created by the simultaneous execution of many jobs. In most modern HPC systems the only shared resource is the parallel file system. We performed no benchmarks for assessing the performance of the parallel file system (mostly SCRATCH with regard to Curie). However, for various reasons we believe that the file system did not show degraded performance while running our tests. First, the number of jobs was relatively small. Because of intra-workflow dependencies at most 120 jobs were running in parallel. Second, the amount of data being processed by a single job was relatively small (about 2 GB). Third, different steps may exhibit different file I/O access pattern, particularly, they may not

Table 6: Runtimes of the steps for selected input data sizes on Curie. Note, step ① is not shown for both use cases because it refers to the sequencer which is not part of our case study implementation. Values separated by a semicolon indicate different inputs. A hyphen indicates a range of values for different reference genomes, alignment parameters and the number of used processor cores.

Step	Use case	Input size (GB)	Threads	Runtime (s)
index (②)	1	0.5; 0.6	1	600; 700
trim sequence (④)	1	0.4; 1.4; 8.4; 17.8	1	45; 210; 2,110; 4,475
align (⑥)	1	1	2-3	3,250 - 16,909
(forward & backward)	1	4	2-3	7,392 - 22,088
	1	17	32	124,269
	1	32.5	32	231,506
align (⑥)	1	8.4	31 - 63	25,342 - 230,862
(forward only)	1	17 - 21	31 - 127	115,770 - 253,761
align (⑥)	1	8.4	31 - 63	25,613 - 97,537
(backward only)	1	17 - 21	31 - 127	184,953 - 252,032
confirm authenticity (⑦)	1	0.5	1	41 - 2,687
	1	2.1	1	207 - 2,819
	1	19.1	1	16,840
	1	37.2	1	31,517
quality check (③)	1	0.4; 1.4; 17	1	30; 110; 1,000
quality check (⑤)	1	0.5; 2; 32.5	1	50; 170; 2,000
index & dictionary (②)	2	0.5	1	500
align (③)	2	11	3	5,500
remove duplicates (④)	2	3.8	1	1,600
index (⑤)	2	3.5	1	65
identify indels (⑥)	2	4	1	1,040
realign indels (⑦)	2	4	1	1,680

read all input data at once and not write all output data at once. Hence, the file I/O workload may be spread over a job's runtime. See, for example, Fig. 3 which suggest such a pattern (although the data in the graph is for memory consumption and the number of active threads). Fourth, the used file system SCRATCH is designed for a bandwidth of up to 150 GB/s. For example, even if all 120 jobs would read input data at once, the file system should be capable to deliver the content in less than two seconds. Even if it takes 10 or 100 fold this time it would be hardly noticeable. If the number of the simultaneously running jobs, their file I/O access pattern or the available bandwidth of the file system change significantly, the scalability may be limited.

5.4. Lessons learned for scaling workflows on large HPC systems

Scaling workflows on large HPC systems is not an easy task, because of the inherent complexity of the workflows, the number of workflow instances to be run, the total size of data to be processed and the complex nature of large HPC systems. Therefore, we recommend a systematic procedure to scale the execution of a large number of workflow instances to a large HPC system.

1. A description of the workflow is needed – similar to the description in Section 2. including a comprehensive list of all needed software packages (cf. Table 1). These information should reveal possibilities for scaling at the level of individual steps and among independent steps of a workflow.
2. Scalability goals need to be defined: how many workflows shall be executed (in total, simultaneously), what is the range for the size of input data (single workflow instance, all workflow instances).
3. Information about the possible target systems' configuration and environment, and their resource management system as well as limits for resource consumption is needed. Table 7 lists different system components and information that is needed for each component.
4. The software packages need to be tested on the target system to verify they function correctly and to determine their resource consumption. If a tool consumes too much resources alternatives may be considered or the developers of them may be asked to improve them. Usually, the vast number of aspects to be considered in the scaling will limit ability of users to tweak specific tools themselves.
5. Selecting or implementing a framework for executing a single workflow instance and orchestrating multiple workflow instances as well as the logistics for transferring data to and from the HPC system. Table 2 shows

Table 7: Information about HPC system components that needs to be gathered for selecting a target system and for orchestrating multiple workflow instances.

Component	Information	Remark
File systems	root path	(user-specific) base of a file system
	quotas (space & files)	Data capacity
	accessibility	May vary on login and compute nodes
	auto removal policy	If and when data gets removed by the system
Compute nodes	number of cores	Limit for exploiting thread-parallelism
	memory	Limit for using shared memory
Software environment	organization	How is the software structured? How are specific software packages used (e.g., by loading modules)?
	compilers	Which compilers are supported?
	runtime environments	Which runtimes (e.g., Java) are supported?
	script interpreters	Which script interpreters (e.g., Perl) are supported?
	task automation	Which means exist to automate tasks (e.g., by shell scripting)?
Job management	job classes	Which job classes are supported? What are their limits (e.g., wall clock time, number of jobs, number of nodes, etc.)?
	job dependencies	Are job dependencies supported?
	minimum job size	Is there a minimum job size (e.g., number of cores/nodes)?
	allocation management	Are there means to determine the use of an allocation?
System access	login	Which means exist for logging into the HPC resource (e.g., ssh, key-based, password-based, gsissh, etc.)?
	data transfer	Means to transfer data to/from the system (e.g., scp, GridFTP)
	access restrictions	Is access to the system limited to certain IPs? Are access means limited to use authorized keys, or are passwords required?
	remote steering	Are protocols for submitting jobs and/or data transfers remotely supported?

an overview of frameworks which may be used to implement the use cases. The decision for a framework may limit the available target systems, and vice-versa.

6. Incrementally implement and run a single workflow. Then, run multiple workflows to observe if any resource consumption limits (number of jobs, runtime, file system quota) are hit. Improve the workflow or the orchestration of several workflows if necessary to achieve the defined scalability goals.

5.5. Remarks for future directions

The main goal of this case study was to study the viability of running genomics workflows on large HPC systems. In the course of the study we implemented two use cases on the PRACE Tier-0 machine Curie (CEA, France).

This section describes what is missing to go into production, and what other aspects may be considered for future developments.

Logistics An important aspect left out in the case study is the logistics, that is, where and when (raw) data is available, how and when it gets staged to the HPC system, how and when its processing is triggered, and how, where and when data products are transferred.

Control flow Some tools will process intermediate data even if the preceding step failed for some reason. Therefore it is necessary to define and to check conditions on the success of a step, and stop the execution of a workflow instance if a condition is not met.

Pool of available resources To enable workflow progress under resource unavailability or high load, and to use the best fitting resource for a specific step, it is needed to consider various resources. Other resources than large HPC systems may include Clouds and Grid-enabled high-throughput clusters.

Ease of use Special emphasis should be put on easy to use interfaces such as the Lifeportal. We believe such interfaces can be made even more user-friendly and efficient in one or more of the following areas.

- The structure of the available tools may be improved – for example, separated into basic, composite, workflows. Rationale: There already exist several hundreds of tools, which make it difficult to keep an overview of.
- The import of large data sets needs to be automated. Rationale: It is cumbersome to import hundreds or thousands of files manually.
- The execution of several hundreds of workflow instances using the same parameter sets on different input data needs to be automated and an overview of the status of them provided. Rationale: Individual items of large scale data sets may be processed with the same workflow. Manually starting them one-by-one would be error-prone.
- A portal should make multiple resources (HPC system, Cloud, Grid) available and let run different workflow steps on different resources (depending on availability, load). Rationale: The used software packages and their requirements are so diverse that not a single resource may satisfy them all equally well.
- A portal shall be connected to an archive, a system where scientific data is preserved for long time. Rationale: For validating and reproducing research results it is required to preserve data and all processing steps that led to them. Because, most if not all that information is available from a portal, it is an ideal place to harvest this data from and use it for the ingest of data into an archive.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-312763. The work was achieved using the PRACE Research Infrastructure resources at Curie, CEA, France as well as the Lifeportal, University of Oslo, Norway.

References

1. The Abel Computing Cluster. `http://www.uio.no/english/services/it/research/hpc/abel/`, [Accessed May 6th, 2014]
2. Centre for Integrative Genetics (CIGENE). `http://www.cigene.no`, [Accessed May 6th, 2014]
3. The Curie supercomputer. `http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm`, [Accessed May 26th, 2014]
4. The Galaxy Project. `http://galaxyproject.org`, [Accessed May 6th, 2014]
5. The Genome Analysis Toolkit (GATK). `https://www.broadinstitute.org/gatk/`, [Accessed May 6th, 2014]
6. Lifeportal – University of Oslo. `http://lifeportal.uio.no/`, [Accessed May 6th, 2014]
7. mapDamage. `http://ginolhac.github.io/mapDamage/`, [Accessed May 6th, 2014]
8. Nofima. `http://www.nofima.no/en`, [Accessed May 6th, 2014]
9. sysstat. `https://github.com/sysstat`, [Accessed May 6th, 2014]
10. Simple Linux Utility for Resource Management (SLURM). `https://computing.llnl.gov/linux/slurm/`, [Accessed May 6th, 2014]
11. A lightweight python module for running computational pipelines. `http://www.ruffus.org.uk`, [Accessed May 23rd, 2014]
12. A scalable bioinformatics workflow engine. `https://bitbucket.org/johanneskoester/snakemake/wiki/Home`, [Accessed May 23rd, 2014]
13. Taverna Workflow Management System. `http://www.taverna.org.uk`, [Accessed May 23rd, 2014]
14. Uniform Interface to Computing Resources. `http://www.unicore.eu`, [Accessed May 23rd, 2014]