A cloud solution for multi-omics data integration

A cloud solution for multi-*omics* data integration

Fabio Tordini*
*Computer Science department
University of Torino – Italy
email: tordini@di.unito.it

***Abstract*—Recent advances in molecular biology and Bioinformatics techniques have brought to an explosion of the information about the spatial organisation of the DNA inside the nucleus. In particular, 3C-based techniques are revealing the genome folding for many different cell types, and permit to create a more effective representation of the disposition of genes in the three-dimensional space. This information can be used to re-interpret heterogeneous genomic data (multi-*omic*) relying on 3D maps of the chromosome.**

The storage and computational requirements needed to accomplish such operations on raw sequenced data have to be fulfilled using HPC solutions, and the the Cloud paradigm is a valuable and convenient mean for delivering HPC to Bioinformatics. In this work we describe a data analysis work-flow that allows the integration and the interpretation of multi-omic data on a sort of "topographical" nuclear map, capable of representing the effective disposition of genes in a graph-based representation. We propose a cloud-based task *farm* pattern to orchestrate the services needed to accomplish genomic data analysis, where each service represents a special-purpose tool, playing a part in well known data analysis pipelines.

***Index Terms*—High performance computing, Cloud computing, Bioinformatics, Systems Biology**

1. Introduction

Since the advent of Next Generation DNA Sequencing technologies (NGS), more and more information about genome organisation becomes available. So much that Genomics laboratories are "caught in a flood of data", and are now facing many of the scale-out issues that High-Performance Computing (HPC) has been addressing for years: raw data resulting from sequencing experiments must be analysed, integrated and archived. These steps pose substantial requirements in speed (in terms of execution time), application scalability and data representation. Orchestrating applications to fulfil these requirements is a complex and delicate task.

Moreover, no data analysis pipeline anywhere in any scientific field works as one monolithic process: different stages of the data analysis process are just fundamentally different, and have different parallelism, memory access and data access requirements. Also, it often makes sense to run the same stage of an analysis in a number of different ways, to demonstrate the robustness of novel results (which are not unusual in fields like Genomics and Bioinformatics), or to tackle different sorts of data, for example one in which a reference genome is available, compared to one where it is not. Here, HPC comes into play: from modelling scientific processes to the use of computers to obtain quantitative results from these models, it turns a domain science into a computational activity.

Luckily enough, the Cloud paradigm has become a consolidated technology that exploits the full potential of virtualised resources to deliver computation and storage via the Internet, enabling a larger resource usage by sharing a given hardware among several users. For instance, cloud computing is wide-spreading in Bioinformatics, just because it is a discipline heavily dependent on data and, even more, on space-consuming and time-consuming data processing tasks [1]. With its various declinations (*DaaS, SaaS, PaaS, IaaS*), the cloud paradigm delivers computational power and storage as dynamically allocated virtual resources, on-demand, relieving scientists from the daunting expense of establishing and maintaining complex computational infrastructures for data processing.

Considering the innovation that high-throughput technologies are introducing to the analysis of genomic data, in this work we propose a custom work-flow for data integration that relies on cloud computing: a cloud solution with a scalable high-performance infrastructure, where software tools for data processing are provided *as services*. By exploiting NuchaRt, a software tool that produces a graph-based representation of the genes placed along the chromosome [2], we can integrate data resulting from Hi-C, RNA-Seq and ChIP-Seq experiments, in order to study the interactions among genetic elements that can reveal insights on biological mechanisms, such as genes regulation, translocations and epigenetic patterns.

The whole infrastructure is characterised by a coarse grain parallelism among software services, and a finer grain that exploits the computing power of shared-memory multi-core architectures. We tested our solution with a use case that presents associated RNA-Seq, ChIP-Seq and Hi-C experiments on the same samples of laboratory mouse [3]. We will show that our cloud task farm helps reducing execution times for running the three data analysis pipelines with the datasets presented in [3].

The rest of this article is organised as follows: in Section 2 we introduce the biological background from which this work has emerged, with a discussion on biological tools and pipelines employed to process data. Also, a brief state of the art of current cloud solutions for Bioinformatics is provided. Section 3 describes our cloud-based task farm solution, presenting the overall infrastructure and explaining the orchestration of multiple pipeline stages as services. Section 4 describes a test case upon which we benchmarked our solution, reporting results in terms of execution time, with a discussion of our achievements. Section 5 concludes this work and highlights some proposals for future works.

2. Background

In this Section we will discuss the background from which this work has emerged, providing a brief overview of the biological and Bioinformatics concepts that motivated the development of our solution. Section 2.2 reports some concrete examples where cloud computing actually brought benefits to the Bioinformatics community.

2.1. Genomic data analysis

The study of chromosome organization in the nucleus of a cell is extremely relevant to gain insights on biological function at the gene level, as well as the global nuclear level, and it will further enable the investigation of pathologies related to genome instability or nuclear morphology [4]. Chromosome Conformation Capture (3C) method and its derivatives measure the frequency at which two chromosome fragments physically associate in a three-dimensional space, based on the propensity for those two locations to become cross-linked together [5]. Among them, the *Hi-C* method exploits NGS techniques to detect those genes that physically interact in the nucleus due to spatial proximity, providing data about the chromosomal arrangement in the 3D space [6].

In previous works we proposed NuchaRt, a tool for Hi-C data analysis that produces a graph-based representation of the genes along the chromosome — a sort of topological map of the chromosome [2], [7]. These maps will be the ground for the integration and analysis of *omics* information (i.e., resulting from different biological experiments), such as RNA-Seq and ChIP-Seq, which can greatly benefit from analysis relying on the 3D maps of the DNA. For instance, mapping point-wise information directly on the chromosome graph allows us to apply spatial statistics, highlighting patterns and correlations between epigenetic features and spatial organization that could not have been detected by considering only the bare proximity on the linear sequence.

2.2. Cloud computing and Bioinformatics

Data are both a blessing and a curse in Bioinformatics: for this reason, it can seriously take advantage from the cloud paradigm, which holds great promises in effectively addressing data storage and analysis problems in Bioinformatics: a typical such analysis often involves downloading data from public sites, installing software tools locally, and running analyses on in-house computer resources. By placing data and software into the cloud and delivering them as services, data and software can be seamlessly integrated into the cloud so as to achieve adequate data storage and analysis.

In the past years, efforts have been made to develop cloud-scale tools, including sequence mapping [8], [9], expression analysis [10], peak caller for ChIP-Seq data [11], and various cloud-based applications for NGS data analysis (such as *BGI Cloud* and *EasyGenomics*. These names can all be seen as examples of the *SaaS* model. Worth to mention is also *Mercury* [12], an analysis pipeline for NGS data that has been deployed in private hardware as well as in the Amazon Web Services via the *DNAnexus* platform.

DNAnexus is an American company that provides an enterprise-focused, API-based *PaaS*, designed to enable clinical and research enterprises to move their analysis pipelines into the cloud, using their own algorithms alongside industry-recognized tools and reference resources, with the goal of creating customized work-flows in a secure, cost-effective, and compliant environment. To the best of our knowledge, remarkable *PaaS* solutions for Bioinformatics are *Eoulsan* — which is a cloud-based platform for high-throughput sequencing analyses — and *Galaxy Cloud* — an open, web-based platform for data intensive biomedical research.

Amazon EC2 represents an example of an *IaaS* model [13], and it offers a variety of VM images provided with a good variety of Bioinformatics tools. Other important examples are Cloud BioLinux [14] and CloVR [15]. The former is a publicly accessible virtual machine for high performance Bioinformatics computing. The latter, instead, is a portable virtual machine for automated sequence analysis. It is also worth to mention that Amazon Web Services (AWS) provides a centralized repository of public datasets, including archives of GenBank, Ensembl, Model Organism Encyclopedia of DNA Elements, Influenza Virus, etc. As a matter of fact, AWS contains multiple public datasets for a variety of scientific fields, from biology to astronomy, chemistry, etc[1]. These datasets are delivered as services (*DaaS*), and can be seamlessly integrated into cloud-based applications [13].

2.3. Data analysis pipelines

Datasets are downloaded from public repositories and analysed using well-known pipelines. The main repository for this kind of data is the Sequence Read Archive (SRA) of the National Centre for Biotechnology Information (NCBI), which makes biological sequence data available to the research community, so as to enhance reproducibility and allow for new discoveries by comparing datasets. In this

1. http://aws.amazon.com/publicdatasets

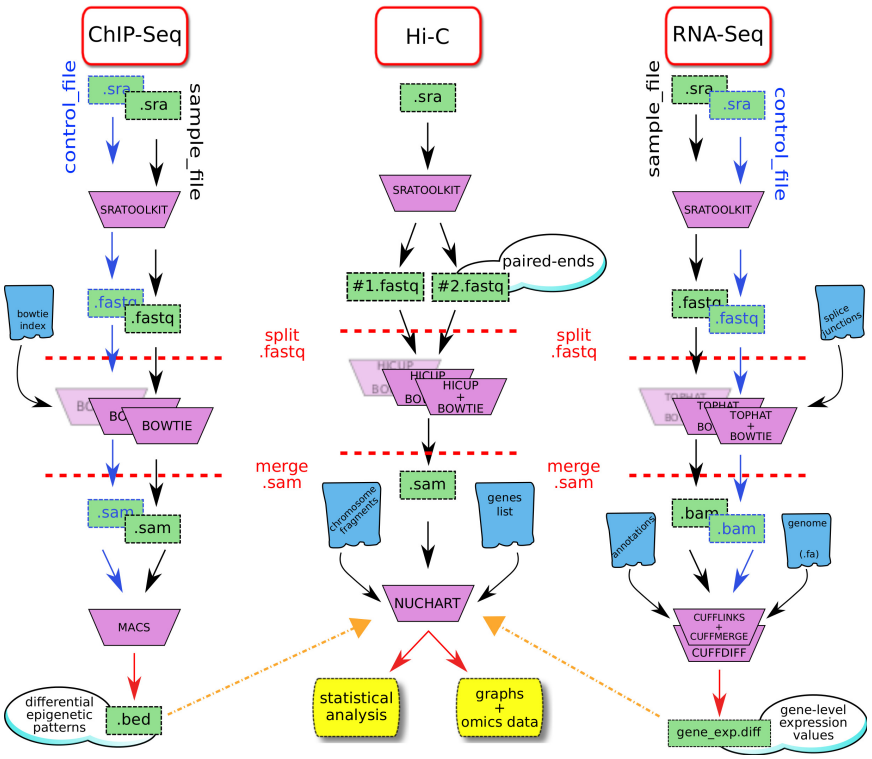

Figure 1. The three Bioinformatics pipelines we are interested on. Red dashed lines indicate a further dataset partitioning, where `.fastq` files are split and processed by many aligners simultaneously. Partial `.sam` files are subsequently merged together to form a definitive output of aligned sequences (see Figure 2)

.

work, we also use this repository as main reference for RNA-Seq, ChIP-Seq and Hi-C experiments.

Two widely used analysis pipelines for RNA-Seq and ChIP-Seq data exist, relying on *Tophat* [16] and *Cufflinks* [17] the former, and on *Bowtie* [18] and *MACS* [19] the latter. Concerning Hi-C data analysis, in [2] and [7] we discussed our approach for the analysis and representation of such data: we refer to those works for better explanations. However, for what it relates to this work, our method relies on *HiCUP* [20] for raw data processing and alignments, and on NuchaRt for generating Hi-C graphs and mapping multi-omic information on the resulting graph.

Each of the three pipelines showed in Figure 1 works on different kinds of data, produced by different biological experiments, conducted with diverse purposes. ChIP-Seq is used to identify those sites where a protein binds to the DNA, while RNA-Seq reveals a snapshot of RNA presence and quantity in a genome. Finally Hi-C produces a catalogue of interacting chromosome fragments, a sort of genome-wide sequencing library that provides a valuable mean for measuring the three-dimensional distances among all possible elements in the genome.

Figure 1 illustrates the stages that compose the three pipelines: raw data resulting from genomic experiments (`.sra` files) are downloaded from public repositories, converted to the `.fastq` format and then aligned against a reference genome. Aligned genomic data is then processed using specific tools.

Dashed lines in Figure 1 indicate a further dataset partitioning: `.fastq` files are split into multiple parts; partial files are processed simultaneously and converted to `.sam` (or `.bam`) file (see Figure 2). All partial results are then merged to form a single aligned file ready to be used in the subsequent steps. Once all data have been processed, we proceed by mapping them onto the Hi-C graph produced by NuchaRt.

Multi-threading is optionally supported by some of the tools listed above. For instance, DNA sequences alignment is a highly parallel task, and the obtainable speedup is significant (though affected by memory overhead). Bowtie allows to execute a parallel search, where threads will run on separate processors/cores and synchronize when parsing reads[2]. Consequently, all tools based on Bowtie (e.g., TopHat and HiCUP) permit to exploit multi-threading during the alignment phase. Among the tools in the Cufflinks suite, *Cuffmerge* and *Cuffdiff* walk through short-reads alignment steps, and they allow to specify the number of threads to be used when performing such operations. NuchaRt is an R tool with an embedded C++ engine, built using high-level parallel programming patterns, that permits to exploit multi-threading parallelism with optimal memory management to achieve remarkable performances during Hi-C data analysis.

2. short DNA sequences

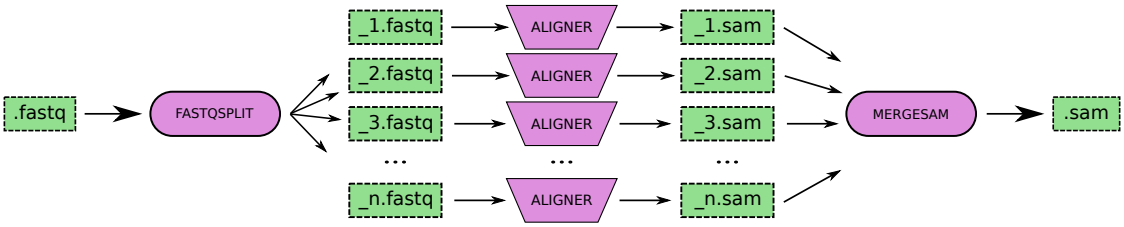

Figure 2. `.fastq` files are split into several parts, according to the available services, and each piece is dispatched to a remote service. Outputs from each service are merged to form a definitive, aligned SAM (or BAM) file used during subsequent steps

3. A cloud-based task farm

Processing genomic data through the pipelines described in Section 2.3 appears manageable, but working on several such data sets within a reasonable wall-clock time frame presents a challenge. Each single step of the pipelines could take up to few hours — depending on the size of input files and on computing capabilities available (see Section 4.1) — and produces results that can be combined together to obtain an enriched data interpretation. If we want our approach to scale genome-wide, we have to take into account that data analysis over such datasets represents a significant computational challenge. Moreover, as the resolution achievable through NGS experiments is increasing and the read-quality data is factored in, a dataset containing full DNA molecules easily reaches to hundreds of gigabytes.

3.1. Set up and communication

In this context, the cloud paradigm represents an appealing solution that permits to obtain large amounts of computing capacity on-demand, with variable pricing. Upon the OpenStack cloud software[3] we built our virtualised infrastructure: OpenStack is a free and open-source cloud operating system, that controls large pools of compute, storage and networking resources, providing an IaaS remote environment for end users and includes computing, networking, storage, and other essential cloud elements, already integrated and interoperable. OpenStack APIs are open-source Python clients, and can run on most existing operating systems, including Linux, Mac OS and Windows. A command-line interface enables to access the platform's API through easy-to-use commands that can be included in scripts to automate tasks.

Beside these technical aspects — which would deserve an extended and more detailed discussion — the rate at which enterprises are adopting open-source technology in their infrastructure continues to grow, and OpenStack technology is one of the fastest growing open-source projects worldwide, which is being deployed by thousands of companies for business-critical workloads and applications. Moreover, one of the advantages of an open-source approach is that it permits to create a much more flexible and vendor-neutral cloud environment.

3. https://www.openstack.org/

We decided to build our basic cloud infrastructure upon OpenStack: we set up a small computing infrastructure composed of a small number of virtual machines (VMs) that will be the computing nodes of our *farm*. Each VM has a direct-attached *ephemeral* storage, plus a secondary persistent storage, and is equipped with commodity multi-core architectures with 2 to 4 virtual CPUs and up to 8GB of RAM. All virtual instances are connected to each other trough a 100Mbps network and run a 64-bit Linux OS.

We decided to deploy a *software-as-a-service* scenario. In this sense, each step of each pipeline is accomplished by a *service* (the trapezoid blocks in Figure 1), and each VM hosts all software need by each pipeline. A VM can execute one service at a time, provided that all needed input is ready and available for the service to run. An additional virtual instance, which we called "work-flow manager" is responsible for orchestrating task scheduling (Figure 3).

To fully leverage the cloud, there are some design aspects that affect architectural choices. Namely, a right combination of data storage, job orchestration and data exchange solutions would help to minimise processing costs. In our scenario, datasets are collected from on-line repositories prior to the start of the analysis. They are accessed in read-only mode, and must be available for all computing instances in order to be used throughout the steps of the pipelines. On the other hand, when a pipeline stage has completed its execution, the resulting output file(s) will be used as input for a (possible) subsequent stage. Consequently, novel produced files need to be accessible by the service that is going to operate on that data.

While the cloud methodology would suggest data migration and replication as the favourite mean of shared storage, we decided that a distributed file system that "ties" together all compute instances is the best solution for our scenario: datasets are huge and need only to be accessed in read-only mode, thus a transferring appears to be unneeded and impractical, also considering the sizes of these datasets. Every virtual instance performs temporary writes on its own local storage, where it has direct I/O access. Writes to the shared folder only happen when a service has completed its task. As for contention, services do not interfere with each other, because each write a single service performs in the shared storage, only affects the files owned and produced by that service. Shared data is periodically backed up on a backup storage.

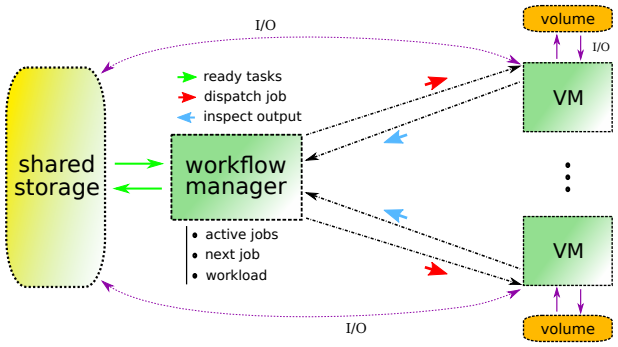

Figure 3. Task farm: task scheduling

3.1.1. Task scheduling. The work-flow manager in Figure 3 mainly organizes the steps of the pipelines into *tasks*[4], which are to be accomplished by services: it maintains a list of active jobs and a list of tasks ready to be scheduled; it is responsible for dispatching jobs to (idle) worker machines; it handles jobs failures and errors.

There are several, well established work-flow management systems (WMS) that have supported Bioinformatics and other scientific work-flows (*Pegasus*, *Taverna* and *Askalon*, among others) which provide several advanced features such as fault-tolerance, task clustering, site selection, resource provisioning, etc. Every WMS has its own pros and cons: OpenStack includes a library named *TaskFlow*, that allows the creation of work-flows where task objects and functions are combined together. Anyway, we found that TaskFlow has limited (or at least confusing) support for orchestrating work-flows over a distributed infrastructure: it permits to declare work-flow engines as *workers*, that are separate processes dedicated for certain task execution. If running on other machines, engines are connected through the *kombu* python messaging library. Differently, Pegasus [21] seems to better suits our needs, likely because it has native explicit support for clusters and grids: it works in combination with *HTCondor*[5], a full-featured workload management system for compute-intensive jobs.

At the time of writing, a full working cloud solution based on Pegasus is still under development, mainly because we started with a small scale cloud infrastructure which was sufficient to test and benchmark our ideas. We have built our own simple WMS, loosely inspired by the above, in order to be able to validate our solution. Simple does not mean less efficient: our WMS reliably schedules tasks marked as *ready* on worker machines that are currently not executing any task. We used the *Paramiko*[6] python library to handle SSH connections to remote machines: with Paramiko we take care of monitoring remote workers, and we developed basic error handling feature that either attempts to retry on a different worker a failed task, or tags it as failed and requests a user-supervised checking.

4. we will use the terms *task* and *job* interchangeably
5. https://research.cs.wisc.edu/htcondor/
6. http://www.paramiko.org/

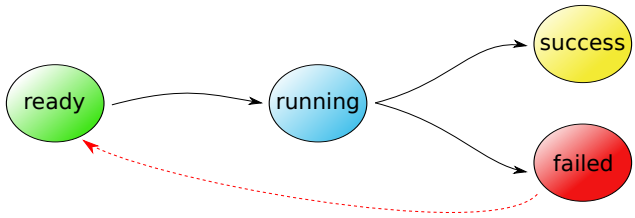

Figure 4. Task states

Each task in our system is a list containing the name of the service (e.g., an application) to be executed, paths to input files, possible parameters needed by the application, and the path to the proper shared output folder.

Tasks are linked together according to a *pattern of execution*. Specifically, for each pipeline in Figure 1, tasks follow a *linear* pattern of execution, because the tasks of a pipeline run one after the other in a serial manner, so as to respect dependencies among tasks. When considering the whole schema from a higher perspective, the overall execution follows an *unordered* pattern, where a set of tasks can be executed in any order, provided their input data is available.

A task life cycle is characterised by different states (Figure 4): a task is *ready* to be fired for execution when all input files it needs are available: when in this state, the work-flow manager dispatches the task to an available service. Once fired, a task is in the *running* state and the work-flow manager updates information concerning VMs' workload, so that jobs scheduling is optimized to minimize the number of idle services. A task moves to the *success* state after it has finished successfully (i.e. no exceptions were raised during running): a successful job has written all its output files into the shared storage, so that its results can be used by a following stage of the pipeline. If a task execution has finished with an error, it enters the *failed* state.

At this point, the failed task is rescheduled, as soon as a worker is ready to execute it. If it fails again it remains in the failed state: due to the nature of the software employed in the stages, a supervised checking is needed in this case, because some input files might be missing or incorrect. A detailed description of the exception occurred is reported, while the pipeline where the failing task belongs is halted.

A coarse grain parallelism is kept up while services are running: every command is sent through a SSH channel, and is in turn managed by a *controller* thread responsible for low-level operations, such as establishing the connection, while the work-flow manager continues its operations. Once the work-flow manager has dispatched a command on a worker machine, the controller thread waits for the command to terminate its execution, and captures the exit status and output messages returned.

3.1.2. Partitioned alignment. Sequence alignment is notoriously a long, time- and resource- consuming task: Bowtie and Bowtie-based alignment tools normally exhibit execution times in the order of hours, depending on dataset size,

TABLE 1. FASTQ FILES SPLIT INTO FOUR PARTS, ALIGNED AND MERGED. EXECUTION TIMES AND SIZES

	full .fastq		split .fastq			
fastq file	**size (GB)**	**align (s)**	**split (s)**	**size (GB)**	**align (s)**	**merge (s)**
SRR206986	3.7	1729	374	0.9	559	149
SRR207094	5.5	4871	571	1.4	1893	276
SRR501780_1	15	28800	1927	3.6	10700	1320

aligning options, computing power available and memory resources. Timings are likely to increase as the size of datasets increases, but this situation is even worst if the physical memory available is too small, causing the operating system to swap pages when memory demands are greater than that physically available for all processes.

Despite being able to exploit thread parallelism during alignment, the size of raw data files is by far the most important factor that influences this execution time. In order to cope with this issue, `.fastq` files can be further split, and alignment can be performed on partial files, while the definitive output is obtained by merging each processed partial file (Figure 2). An ideal policy would suggest to create as many partial files as there are working machines available: the more the parts, the smaller the files, thus the faster should be the alignment. In reality, there is a trade-off among the number of parts, splitting time, alignment time and merging plus sorting time.

In our task farm each computing resource performs the alignment on a partial `.fastq` file, yielding a partial `.sam` (or `.bam`) file of aligned sequenced reads. Partial outputs are then merged together and sorted.

Table 1 reports some details concerning timings and file sizes during alignment steps for the ChIP-Seq (first row), RNA-Seq (second row) and Hi-C (third row) pipelines. First column on the left reports experiments names, as they have been downloaded from the NCBI repository[7].

Timings reported are in seconds and reflect wall-clock time measured using Linux `time` command, on virtual instances equipped with 2 vCPUs. Alignment time for split files is the average of the four alignments, executed concurrently by four services. The last column on the right reports the sum of the merge and sort operations. These timings could be lower if more computing power was available (i.e., more CPUs) for multi-threading parallelism with aligner tools.

This solution is a valuable mean for reducing alignment timings — compared to the processing of a whole dataset, maintaining the same configuration — but mostly it allows to distribute the workload over several computing instances, making effective use of the IaaS cloud paradigm. Nevertheless, performance is still heavily dependent on the underlying computing capabilities and physical memory available. Also, we used basic configurations when launching alignment tools, but the number of options varies for each tool, which permit to customise the process and obtain more accurate and detailed outcomes, at the cost of higher memory consumption.

7. Experiments from [3] are hosted at http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE29184

4. Test Case

We tested our infrastructure using datasets from the work of Shen et al. [3], which is particularly interesting since it presents associated RNA-Seq, ChIP-Seq and Hi-C experiments on the same samples of laboratory mouse. In detail, the work comprises 143 datasets from 19 different tissues of mouse, and is aimed at finding those genomic regions that affect genes operation and behaviour. The peculiarity of this work is the concomitant presence of Hi-C data, which allowed scientists to demonstrate that the mouse genome is organized into domains of regulated enhancers and promoters.

4.1. Computational costs

The following tables summarize the computational costs encountered while running our task farm over some of the Shen et al. experiments [3]. Our task farm was implemented over a cloud infrastructure built on top of OpenStack, as described in Section 3, with a farm of 4 virtual instances plus a controller node, where the work-flow manager was running. All instances had a very simple, identical configuration, with 2 vCPUs and 8GB of RAM.

Tables show timings (in seconds) and input sizes for each step of each pipeline, for both the control file (marked with a '*C*') and the sample file (marked with a '*S*'). A control file normally refers to data from an experiment conducted on a healthy cell, while the sample file contains data from a diseased cell. Both the ChIP-Seq pipeline and the RNA-Seq pipeline need to be executed for both control and sample files, in order to identify significant features resulting from comparing the two experiments. When both files are processed together (as it is the case with MACS or Cuffdiff), timings are reported in the sample column only, while input sizes are showed for both.

Tools employed in each step might output more than one file, sometimes containing statistics on the alignment or a summary of the executed operations. However, input sizes reported in our tables reflect the sum of the files (if more than one) used as input by a specific task. Also note that alignment timings reported are the sum of the average split time, average aligning time and average merge (and sort) time.

For what it concerns ChIP-Seq, it is the less demanding pipeline: smaller datasets led to reduced memory overhead

TABLE 2. EXECUTION TIMES AND SIZE OF INPUT FILES FOR EACH STEP OF THE CHIP-SEQ PIPELINE

	ChIP-Seq			
	SRR206986 (*S*)		SRR206994 (*C*)	
step	**time (s)**	**input (GB)**	**time (s)**	**input (GB)**
convert_fastq	138	0.49	149	0.51
alignment	982	3.7	1017	3.9
MACS	344	2.5	–	2.6

TABLE 3. EXECUTION TIMES AND SIZE OF INPUT FILES FOR EACH STEP OF THE RNA-SEQ PIPELINE

	RNA-Seq			
	SRR207094 (*S*)		SRR207095 (*C*)	
step	**time (s)**	**input (GB)**	**time (s)**	**input (GB)**
convert_fastq	171	0.45	240	0.53
alignment	4871	5.5	4006	6.2
cufflinks	3156	0.6	3421	0.8
cuffmerge	287	1.2	–	1.3
cuffdiff	18840	2.5	–	–

and faster execution time. Table 2 reports details of a run using SRR206986 experiment as sample file, and SRR206994 experiment as control file (the latter was an input cortex cell line, while the former had CTCF cortex data). Aligned chromosome reads from both control and sample files are used together as input for the MACS tool, which evaluates the significance of enriched genomic regions where proteins bind to the DNA, and generates a simple textual output containing genomic locations, specified by chromosome, begin and end positions, and some more optional information, including the *p*-value computed for each identified region.

RNA-Seq pipeline is slightly more complex, and more steps are required to interpret differential gene expression. TopHat uses Bowtie for DNA sequences mapping and alignment, and builds a database of needed and unneeded genomic features involved in genes expression. This specific phase is very time- (and memory-) consuming, and there is little room to speed it up because it does not benefit from multi-threaded execution. , which is instead exploited when aligning reads with Bowtie. When we split `.fastq` files we can distribute the workload over several working machines, attempting to reduce the overall execution time — on average, splitting the file into 4 parts and aligning each part on separate instances, halves the wall-clock execution time, with respect to processing the full dataset — but the overhead of merging partial outputs and sorting the definitive one is not negligible.

At the Cufflinks stage, sample and control libraries are compared, in order to quantify genes actual expression: Cuffmerge and Cuffdiff are responsible for merging and comparing expression levels of genes in both control and sample RNA-Seq experiments. The differential expression at the gene level is reported in a file named `gene_exp.diff`.

The Hi-C pipeline does not use a control file, but operates on raw Hi-C data, contained in the `.sra` file, which is converted into two `.fastq` files (with forward and reverse spot respectively, due to the nature of the experiment). These files are split, as explained above, processed using HiCUP that yields `.sam` files containing pairs of chromosome fragments which are likely to be close to each other in a 3D view of the DNA.

The Hi-C pipeline proceeds by constructing a graph from the `.sam` file produced by HiCUP using NuchaRt: in these graphs nodes represent genes and edges link two genes which are found to be close to each other, according to the `.sam` file. Each pair of genes linked by an edge is likely to interact and influence each other's behaviour. While building a whole-genome graph would be technically possible with NuchaRt, we normally focus on some genes of interest, and rather build a *neighbourhood* graph for these genes [7].

Upon these graphs we map the heterogeneous information obtained from the other pipelines, and then proceed by rendering the resulting network and performing statistical analysis over it, by just drawing from the statistical libraries set provided by the R environment.

The execution time for the NuchaRt phase is relative to the graph built for few genes of interest (SOX2, POU5F1, REST) on SRR501780 experiment. The reported execution time is the sum of 5 distinct steps (as described in [2]).

Minor considerations. Notably, long-lasting tasks are more likely to incur in platforms or connection errors, causing the abortion of the failing job and which would force a job resubmission, increasing the total execution time. Our task farm can handle failures by trying to resubmit a task in a different working instance, but has little automated control over platform errors: misconfiguration of user software or missing libraries can impede an application to start; huge datasets can saturate physical memory, causing an application to immediately abort. As soon as we upgrade our infrastructure with a state-of-the-art WMS, we foresee better handling of failures and errors.

It might also be the case that some stages' execution time largely exceed the others, causing a pipeline to halt, waiting for a output data to be ready. In our case, this happened with the Hi-C pipeline, where the NuchaRt stage was waiting for the outputs of both ChIP-Seq and RNA-Seq before it could build graphs with mapped multi-omic information.

5. Conclusion

The explosion of experimental datasets available for genome analysis and multi-omic data integration will pose difficult challenges of data management, which should be carefully considered by computer scientists working in the field. Cloud computing can be very useful for this kind of analysis, since the on-demand paradigm is well suited with the possibility (and necessity) of providing computational resources that can host a variety of different workloads, and that can be deployed and scaled-out through the rapid provisioning of additional virtual resources as soon as new experimental data becomes available.

TABLE 4. EXECUTION TIMES AND SIZE OF INPUT FILES FOR EACH STEP OF THE HI-C PIPELINE

	HiC			
	SRR501780_1		SRR501780_2	
step	**time (s)**	**input (GB)**	**time (s)**	**input (GB)**
convert_fastq	1517	3.0	1624	3.2
alignment	13606	15+15	–	–
NuchaRt	387	∼8	–	–

The presented cloud-based task farm solution attempts to provide an optimal exploitation of the available resources, and permits to produce results in a standardized and reproducible way. In our vision, this solution could be the background for a Cloud platform that provides Bioinformatics scientists with computing facilities, software tools and easy access to public data repositories. What we expect is a complete solution for dealing with complex scientific workflows modelled as pipelines, where software services are made available in an fully equipped environment that can embrace novel data processing and visualisation techniques, as soon as novel discoveries spring out from Genomic communities.

Acknowledgment

This work has been partially supported by the EC-FP7 STREP project "REPARA" (no. 609666) and the EU H2020 "RePhrase" project (no. 644235). The author also wishes to thank Ivan Merelli for his support on biology-related topics, and prof. Marco Aldinucci for the fruitful discussions and teachings.

References

[1] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Cloud and heterogeneous computing solutions exist today for the emerging big data problems in biology." *Nature reviews. Genetics*, vol. 12, no. 3, p. 224, Mar. 2011.

[2] F. Tordini, I. Merelli, L. Milanesi, P. Liò, and M. Aldinucci, "NuchaRt: embedding high-level parallel computing in R for augmented Hi-C data analysis," in *Post-Conference proceedings of the 12th Intl. meeting on Computational Intelligence methods for Bioinformatics and Biostatistics (CIBB 2015)*, ser. LNBI. Springer, 2016, to appear.

[3] Y. Shen, F. Yue, D. F. McCleary, Z. Ye, L. Edsall, S. Kuan, U. Wagner, J. Dixon, L. Lee, V. V. Lobanenkov, and B. Ren, "A map of the cis-regulatory sequences in the mouse genome," *Nature*, vol. 488, no. 7409, pp. 116–120, Aug. 2012.

[4] J.-M. Belton, R. P. McCord, J. H. Gibcus, N. Naumova, Y. Zhan, and J. Dekker, "Hic: A comprehensive technique to capture the conformation of genomes," *Methods*, vol. 58, no. 3, pp. 268 – 276, 2012, 3D chromatin architecture.

[5] J. Dekker, K. Rippe, M. Dekker, and N. Kleckner, "Capturing Chromosome Conformation," *Science*, vol. 295, no. 5558, pp. 1306–1311, Feb. 2002.

[6] E. Lieberman-Aiden, N. L. van Berkum, L. Williams, M. Imakaev, T. Ragoczy, A. Telling, I. Amit, B. R. Lajoie, P. J. Sabo, M. O. Dorschner, R. Sandstrom, B. Bernstein, M. A. Bender, M. Groudine, A. Gnirke, J. Stamatoyannopoulos, L. A. Mirny, E. S. Lander, and J. Dekker, "Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome," *Science*, vol. 326, no. 5950, pp. 289–293, Oct. 2009.

[7] F. Tordini, M. Drocco, C. Misale, L. Milanesi, P. Liò, I. Merelli, and M. Aldinucci, "Parallel exploration of the nuclear chromosome conformation with NuChart-II," in *Proc. of Intl. Euromicro PDP 2015: Parallel Distributed and network-based Processing*. IEEE, Mar. 2015.

[8] T. Nguyen, W. Shi, and D. Ruden, "Cloudaligner: A fast and full-featured MapReduce-based tool for sequence mapping," *BMC Research Notes*, vol. 4, no. 1, pp. 1–7, 2011.

[9] M. C. Schatz, "Cloudburst: highly sensitive read mapping with MapReduce." *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009. [Online].

[10] L. Zhang, S. Gu, Y. Liu, B. Wang, and F. Azuaje, "Gene set analysis in the cloud." *Bioinformatics (Oxford, England)*, vol. 28, no. 2, pp. 294–295, Jan. 2012.

[11] X. Feng, R. Grossman, and L. Stein, "Peakranger: A cloud-enabled peak caller for ChIP-Seq data." *BMC Bioinformatics*, vol. 12, p. 139, 2011.

[12] J. G. Reid, A. Carroll, N. Veeraraghavan, M. Dahdouli, A. Sundquist, A. English, M. Bainbridge, S. White, W. Salerno, C. Buhay, F. Yu, D. Muzny, R. Daly, G. Duyk, R. A. Gibbs, and E. Boerwinkle, "Launching genomics into the cloud: deployment of Mercury, a next generation sequence analysis pipeline," *BMC Bioinformatics*, vol. 15, no. 1, pp. 1–11, 2014.

[13] V. A. Fusaro, P. Patil, E. Gafni, D. P. Wall, and P. J. Tonellato, "Biomedical cloud computing with Amazon web services," *PLoS Comput Biol*, vol. 7, no. 8, pp. 1–6, 08 2011.

[14] K. Krampis, T. Booth, B. Chapman, B. Tiwari, M. Bicak, D. Field, and K. E. Nelson, "Cloud biolinux: pre-configured and on-demand bioinformatics computing for the genomics community," *BMC Bioinformatics*, vol. 13, no. 1, pp. 1–8, 2012.

[15] S. Angiuoli, M. Matalka, G. Gussman, K. Galens, M. Vangala, D. Riley, C. Arze, J. White, O. White, and W. Fricke, "Clovr: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing." *BMC Bioinformatics*, vol. 356, no. 12, Aug 2011.

[16] C. Trapnell, L. Pachter, and S. L. Salzberg, "Tophat: discovering splice junctions with rna-seq," *Bioinformatics*, vol. 25, no. 9, pp. 1105–1111, 2009.

[17] C. Trapnell, B. A. Williams, G. Pertea, A. Mortazavi, G. Kwan, M. J. van Baren, S. L. Salzberg, B. J. Wold, and L. Pachter, "Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation," *Nature Biotechnology*, vol. 28, no. 5, pp. 511–515, May 2010.

[18] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, p. R25, 2009.

[19] Y. Zhang, T. Liu, C. A. Meyer, J. Eeckhoute, D. S. Johnson, B. E. Bernstein, C. Nusbaum, R. M. Myers, M. Brown, W. Li, and X. S. Liu, "Model-based Analysis of ChIP-Seq (MACS)," *Genome Biology*, vol. 9, no. 9, pp. R137+, Sep. 2008.

[20] Babraham Bioinformatics, Available from: http://www.bioinformatics.babraham.ac.uk/projects/hicup/, 2012, Accessed: 2016-05-20.

[21] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Gener. Comput. Syst.*, vol. 46, no. C, pp. 17–35, May 2015.