# Exploiting Docker containers over Grid computing for a comprehensive study of chromatin conformation in different cell types

# Exploiting Docker Containers over Grid Computing for a comprehensive Study of Chromatin Conformation in different cell types

Ivan Merelli[a], Federico Fornari[d], Fabio Tordini[b], Daniele D'Agostino[c], Marco Aldinucci[b], Daniele Cesini[d]

[a]*Institute for Biomedical Technologies, National Research Council of Italy, Segrate (MI), Italy*
[b]*Computer Science Department, University of Torino, Italy*
[c]*Institute for Applied Mathematics and Information Technologies "E. Magenes", National Research Council of Italy, Genoa, Italy*
[d]*CNAF Section - Italian Institute for Nuclear Physics, Bologna, Italy*

**Abstract**

Many bioinformatic applications require to exploit the capabilities of several computational resources to effectively access and process large and distributed datasets. In this context, Grid computing has been largely used to face unprecedented challenges in Computational Biology, at the cost of complex workarounds needed to make applications successfully running. The Grid computing paradigm, in fact, has always suffered from a lack of flexibility. Although this has been partially solved by Cloud computing, the on-demand approach is way distant from the original idea of volunteering computing that boosted the Grid paradigm. A solution to outpace the impossibility of creating custom environments for running applications in Grid is represented by the containerization technology. In this paper, we describe our experience in exploiting a Docker-based approach to run in a Grid environment a novel, computationally intensive, bioinformatic application, which models the DNA spatial conformation inside the nucleus of eukaryotic cells. Results assess the feasibility of this approach in terms of performance and efforts to run large experiments.

*Keywords:* Grid Computing, Docker Containers, Data modelling, Chromatin Conformation, Computational Biology

## 1. Introduction

In the last decade, Grid infrastructures had an important role in supporting the biomedical scientific community to perform its compute-intensive analysis, finding innovative solutions for reducing computational times [1]. Nowadays, due to the exponential availability of new biological data and considering the increasing complexity of bioinformatic applications, Big Data-like situations and computational demanding problems are becoming more and more urgent in

Computational Biology [2]. Distributed infrastructures, such as those based on the Grid and Cloud paradigms, represent one of the possible solutions to achieve the required amount of computational power in a cost-effective way [3].

Grid computing can be defined as a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. Many years after the spreading of this technology, we can say that Grid is a wrapper to freely access remote multi-institutional resources, e.g. with the EGI Foundation, formerly named the European Grid Infrastructure [4], for research activities. On the other hand, Cloud computing has been defined as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. In contrast with the definition of Grid, we can define Cloud computing as a way to deliver on-demand, scalable, pay-per-use services, which rely on virtualized computational facilities hosted by single institution, over Internet.

Above all, Cloud infrastructures can be used for business, commercial and research purposes, but the idea is always to have on-demand resources, working on a pay-per-use basis. On the other hand, Grid platforms are mainly used for research by Virtual Organizations, that are a dynamic sets of individuals and/or institutions that share a goal to be pursued using the Grid resources, but for free. While the users of a Cloud infrastructure are customers, the users of a Grid are members of one or more virtual organizations. Working for public bodies, we believe that the Grid computing paradigm is still important for sharing resources, in particular in the context of large consortia oriented to specific research purposes.

However, Grid-based solutions are not completely satisfactory to create reliable computing environments for complex tasks [5, 6]. A key issue is that a Grid offers poor chances to customise the computational environment. For example, it is quite common in Computational Biology to make use of relational databases and/or web-oriented tools to perform analyses, store output files and visualise results, which are difficult tasks to exploit without having administration rights on the used resources. Another problem concerns the huge amount of bioinformatic packages available in different programming languages and frameworks (such as R, Perl and Python) that typically require many dependencies and fine-tuned customisations of the settings.

These are the reasons why Docker [7] represents a very appealing solution to create a custom environment over Grid [8]. In this paper, we describe our experience in exploiting a Docker-based approach for a bioinformatic application, which basically consists in modelling the DNA spatial organization, i.e. the chromatin conformation, in the nucleus of different cell types [9]. Although many experiments to study the chromatin conformation are daily performed in molecular biology laboratories all around the world [10, 11, 12], the interpretation and modelling of this data is still a complex and challenging task [13, 14].

In particular we exploited a novel experimental technique, called *chromosome conformation capture* (3C) [15], which allows to measure the proximity of DNA strands in the nucleus. Among novel 3C-based methods, *Hi-C* uses Next-Generation Sequencing (NGS) techniques to interrogate 3C experiments more comprehensively and with an increased throughput [16]. The "Hi" thus stands for "high-throughput", and sometimes it is written as *HT-3C*. The data produced by a Hi-C experiments are coupled reads that describe the frequency by which two strands of DNA are close each other inside the nucleus. The output of a Hi-C analysis is a list of coupled locations along chromosomes, which can be represented as a *contact map*, i.e. a square matrix $Y$ where $Y_{i,j}$ stands for the sum of read pairs matching, respectively, in position $i$ and position $j$.

Contact maps are reliable while looking at the intensity of the interactions inside a chromosome or between two chromosomes, but become unsuitable to depict the neighbourhood of genes, as they lack the possibility to define metrics for computing distances between two or more genes. On the other hand, graph-based models of Hi-C data can be very useful for creating representations where other *omics* data can be mapped, in order to characterise different spatially associated domains [17, 18]. By exploiting their higher level of expressiveness, graphs permit the integration of multi-omic data and facilitate their statistical analysis [19, 20].

Therefore our goal is to exploit a Grid computing infrastructure for creating a collection of graph-based models relying on a set of publicly available Hi-C experiments performed in different laboratories worldwide, in order to produce a catalogue on which new results from different multi-omic experiments can be modelled, analysed and interpreted. To accomplish this task a huge computational power is needed, and Grid computing perfectly suites our requirements, if a proper execution environment can be created.

This paper is organised as follows: Section 2 discusses Related Works and Section 3 provides a background on the biology-related topics of the application we considered. Section 4 gives an overview of the computational infrastructure we exploited, Section 5 describes how the computation was performed, while Section 6 discusses the achieved results. Section 7 concludes this work.

## 2. Related works

The design and effective deployment of reliable, scalable and portable complex computational systems is a major issue in many research and industrial fields [21, 22]. This is the reason why there have been significant progresses in building virtualization layers for operating systems and, more recently, software applications [23, 24]. A first approach for providing such results is represented by the full virtualization technology [25], in which each virtualized system gets its own subset of physical resources, with a minimal sharing and an high level of isolation with respect to other systems running on them. Examples of platforms supporting it are Xen [26], KVM [27] and VMware [28]. A second approach is represented by the use of containers [29], which provide a lower level of isolation, but the resulting lightweight overhead allows easily to run thousands of instances on a host, result that is almost impossible with the full virtualization [30, 31].

The most popular example of the container technology is the Docker platform [7], which allows for the creation and configuration of software containers for deployment on a range of systems [32]. However, other solutions are available, such as rkt [33] from the Linux distributor CoreOS, LXD [29] from Canonical Ltd. - which is the company behind Ubuntu, or OpenVZ [34], which is an extension of the Linux kernel. Behind these core architectures, a number of orchestrator tools are available. Among the most popular solution is worth citing Kubernetes [35], a project released as open source by Google, or Apache Mesos [36]. Due to the large interest in the container technology, the most popular Cloud providers have released solutions for dealing with systems relying on it, which are often defined Container applications as a Service - CaaS [37]. An example is Amazon AWS ECS [38], a manager of Docker images, which can store images in the accompanying ECS Registry, run Docker containers (ECS Runtime) and schedule / orchestrate / monitor these container instances (AWS CloudWatch). Microsoft Azure Container Service (ACS) works together with Docker and Apache Mesos as container orchestration engine [39]. Rancher also supports Docker Swarm, Kubernetes and Apache Mesos [40].

Obviously there are pros and cons for both using the full virtualization or containers. The most important aspects are represented by security and performance. As said before, if a system needs a full isolation and a set of guaranteed resources, the full virtualization is the proper solution. On the contrary if it is sufficient to isolate just processes and the purpose is to serve a large number of them at a time, then containers have to be adopted. Moreover, a full virtualized system usually takes minutes to start, whereas containers take seconds.

While the use of full virtualization represents a consolidated approach in Bioinformatics [41, 42], in the last few years several studies have begun to focus on the use of Docker [43, 44, 45, 46] and a repository of Docker images for Biomedical applications, called BioShaDock, has also been developed [47]. Considering the use of Docker in distributed computing environments for Biomedicine, some attempts have been done to combine Docker Swarm with OpenStack [48] and to develop containerized Bioinformatic services on Cloud [49].

Although just a few attempts have been done to run container on Grid, the resources available on Cloud infrastructures for researchers (such as the EGI Federated Cloud) are still not comparable with the resources available on the Grid. Moreover, several communities still have computing models based on the Grid platform and hence exploiting user-level containerization techniques can increase the flexibility without re-creating from scratch their computing models.

Considering the Desktop/Edge Grid approach, some implementations based on Docker in combination with BIONC, the most popular middleware for volunteer computing, are available [50, 51]. Because of the small footprint of Docker, the general outcome of these experiments is that using container over Desktop/Edge Grid provide good performance giving the applications much more agility [52]. Considering more traditional service based Grids, IBM has also worked to get Docker containers running on the top of its Platform LSF scheduler and Adaptive Computing has just updated its Moab scheduler with Docker support [53]. Moreover, Docker images are available for Selenuium Grid [54], and a Docker compliant version of HTCondor has been released by CERN [55].

However, to the best of our knowledge, no extensive experiments have been done or at least reported using these infrastructures. The only published application that actually combines Docker, more precisely the *Udocker* version, and Grid has been developed in the field of biomedical image processing [56]. This work presents a sort of parameter sweep application to identify the best parameters for the analysis of computer tomography data. From the technological point of view, the tests compare two approaches for managing the software dependencies of the code, i.e. the use of a Storage Element for the software libraries and the use of containers for executing the jobs. Considering the similarities of this work with our approach, we will compare our results with their ones at the end of Section 6.1.

## 3. Bioinformatic background

Recent advances in high throughput molecular biology techniques and Bioinformatics have provided insights into chromatin interactions on a genome scale [16]. These techniques allow the description of the nucleus organization at unprecedented resolution, offering the possibility to study the structural properties and spatial organization of chromosomes. This is of critical importance for understanding and evaluating the regulation of gene expression, DNA replication, repair and recombination [57].

In particular, Hi-C experiments, which are the combination of high-throughput sequencing with 3C techniques, allows the characterization of long-range chromosomal interactions. They give in fact information about DNA fragments that are cross-linked together due to spatial proximity. This is achieved by reading their blunting ends through a special protocol called paired-end sequencing (see Figure 1), which provides a map containing the contact probability data that describe the chromosomal organization in the 3D space of the nucleus.

The resulting *contact map* reports the contact frequencies between a group (or groups) of genomic bins. The contact frequency between two bins relies on their spatial proximity, and it is expected to reflect their actual physical distance. Despite contact maps are reliable while looking at the intensity of the interactions inside a chromosome or between two chromosomes, they becomes unsuitable to depict the neighbourhood of a gene, as they lack the possibility to define a metric for computing the distance between two or more genes.

Graphs have a higher level of expressiveness, since nodes represent the actors of a process while edges identify relationships among the actors, and graph-based model of Hi-C data can be very useful for creating a representation where other *omic* data can be mapped. Structural properties of a graph can reveal significant information on how the actors of the represented process interact, while parallel algorithms can be employed to operate over a graph.

Some tools have been proposed to analyse Hi-C data using a graph-based representation, among which the most popular is probably NuChart [58, 59].

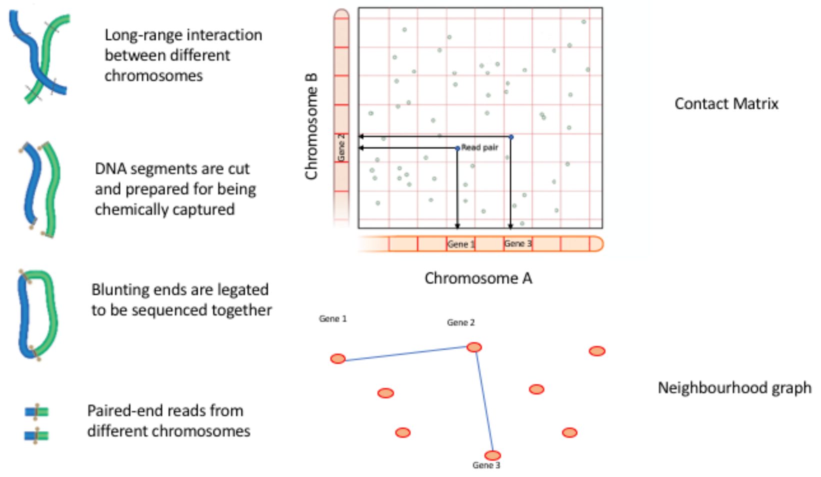
Long-range interaction between different chromosomes
DNA segments are cut and prepared for being chemically captured
Blunting ends are legated to be sequenced together
Paired-end reads from different chromosomes
Chromosome B
Gene 2
Read pair
Gene 1
Gene 3
Chromosome A
Contact Matrix
Gene 1
Gene 2
Gene 3
Neighbourhood graph

Figure 1: The workflow of a typical Hi-C experiment. DNA strands close to each other in the 3D space of the nucleus are fragmented and cross-linked using molecular biology techniques. Then, using paired-end sequencing, couples of fragments are identified as co-localized in the nucleus of cells and the corresponding bin count in the contact matrix is updates consequently. The contact matrix is converted into a graph-based representation to enable more complex analysis.

Inspired by web applications such as Google Maps, NuChart is an R package that elaborates Hi-C information to provide a Systems Biology oriented, gene-centric view of the three-dimensional organization of the DNA in the nucleus. NuChart can be used to describe the DNA conformation in the neighbourhood of selected genes by mapping on the achieved graph genomic features that are important for controlling gene expression at epigenetic level and of multi-omic data on the nodes, thus facilitating statistical analysis [19, 20].

## *3.1. Algorithm for chromatin conformation analysis*

We recall that a graph $G(V, E)$ is a formal mathematical representation of a collection of vertices $(V)$, connected by edges $(E)$ that model a relationship among vertices. In this context, vertices represent genes (e.g. an ordered set of an organism's genes) labelled with gene names. Here we define paired-ends Hi-C reads as a connection meaning a spatial relationship between two genes (see Figure 1). It follows that two genes $g_1, g_2 \in V$ are connected if there exists a connection encompassing both of them. Connections corresponds to the edges of the graph, i.e. $e = (g1, g2) \in E$.

The neighbourhood graph $N_G(V', E', w)$, $N_G \subseteq G$, can be defined as an undirected weighted graph where:

- $V' \subseteq V$ is a set of genes;
- $E' \subseteq E$ is a set of existing edges;

- $w : E \rightarrow [0,1]$ is a function that assigns a probability of actual physical proximity for each pair of adjacent genes $(g_i, g_j)$ connected by means of a paired-ends Hi-C read.

The neighbourhood graph is obtained by starting from a given root vertex $v$, which includes all vertices adjacent to $v$ and all edges connecting such vertices, including the root vertex. With these premises, our neighbourhood graph represents a topological map of the specific nucleus region to which a gene belongs.

### *3.1.1. Graph construction*

The construction of such graphs is based on the exploration of static datasets: raw data resulting from Hi-C experiments are processed through the HiCUP pipeline [60], which produces millions of paired-end *reads* (i.e., short DNA sequences with start/end coordinates) written into Sequence Alignment Map (SAM) files. These reads represent the main input of NuChart, because they expose the spatial information exploited by the process to infer a topological structure of the DNA.

NuChart evaluates reads against a reference genome that contains the coordinated chromosome fragments, and a list of genes with their positions (again, coordinates) along the DNA. The basic mechanism in the exploration stage loops over the set of desired initial genes: for each gene, it looks for all connections (i.e. edges) $e_i \in E$ (paired-ends Hi-C reads) whose first *end* encompasses the current gene – basically comparing chromosome name and coordinates. Among the found connections, it searches for neighbouring genes that might be located within $c$'s second *end*. The reason for searching adjacent genes in a read's second end come from the way in which Hi-C (and 3C-based) experiments are conducted: Hi-C identifies spatially adjacent DNA segments in terms of three-dimensional space. If a gene is found on a read's first end, a possible gene found in the second end is likely to be spatially adjacent, unless there are sequencing errors and biases.

If we define the root of our neighbourhood graph to be at level 0, a search at level 1 yields all the genes directly adjacent to the root. It follows that a search at level $i$ returns all genes directly adjacent to any gene discovered at level $i-1$, starting from the root. The final graph is returned in form of a list of edges linking couples of nodes.

The graph exploration proceeds according to a Breadth First Search (BFS) strategy: starting from one or more root genes (the starting node(s) of the graph), it expands the discovered graph one level at a time, until either all the reachable nodes have been found, or up to a chosen distance from the root. The BFS-like graph exploration results in a data-parallel procedure, in which any arbitrary subset of reads can be processed independently from each other, provided that no data dependency is involved in their manipulation. Ideally, it can be parallelized in a seamless way by just taking the kernel of the procedure and putting it into a parallel loop pattern [61].

The graph exploration has been organised in a level-synchronised way, and concurrent write accesses to data structures shared between worker threads have been managed. For example, each iteration of the loop builds a local graph, and a mechanism of graph merging from local graphs to a global output graph (actually one for each level) has been provided. Globally, this approach amounts at providing a reduce phase after each loop step, in which per-thread local data structures are merged into per-level global ones.

### *3.1.2. Normalization*

Hi-C readouts generally suffer from data biases that may lead to false-positives or false-negatives during data analysis. These biases might result from a number of sources, including sequencing machines' precision and read alignment slips, while some might be specific to the Hi-C experiment protocol. In NuChart, particular attention is given to the detection and normalisation of systematic biases, in order to correct them and avoid wrong data interpretation [62].

In our vision, an edge identifies the existence of an Hi-C read that encompasses two connected genes: normalising each edge using genomic features – which may include the DNA sequence, genes and gene order, regulatory sequences and other genomic structural landmarks – yields a significance estimate of fragment interactions. Such an estimate is then used as the *weight* of the edge, which assumes the role of likelihood of physical proximity for the involved genes.

For each edge, a contact map ($Y$) is constructed that directly models the read count data at the defined resolution. A Hi-C data matrix is symmetrical, and thus we consider only its upper triangular part, where each point $i, j$ of $Y$ denotes the intensity of the interaction between positions $i$ and $j$. Using the local genomic features that describe the chromosome, we can set up a Generalised Linear Model (GLM) with Poisson regression, with which we estimate the maximum likelihood of the model parameters.

The model is given by the formula: $e(Y) = gX^T\beta$. Here $Y$ is the dependent variable, or rather the contact map that contains the measured contact frequencies: the assumption of this GLM is that the measured interaction frequencies are generated from a particular distribution in the exponential family, the Poisson distribution in our case, which is used to count the occurrences in a fixed amount of space. $X$ is the independent variable, which is built from chromosome length and Guanine Cytosine content, measured for each locus of the contact map. $\beta$ denotes the parameter vector to be estimated: $X^T\beta$ is thus the linear predictor, that is the quantity which incorporates the information about the independent variables into the model. It is related to the expected value of the data through the link function $g$.

The maximum likelihood estimate for each edge is computed using the Iteratively Weighted Least Squares algorithm (IWLS) [63]. The best-fit coefficients returned by the linear regression are used to compute the final score of an edge, so that the edge contains an estimate of the physical proximity between the two genes it links, plus the genomic information for both linked genes. The regression is run until a convergence criterion is met: in our case, we check that the absolute value of the $\chi^2$ (chi-squared) difference at each iteration is less than a certain threshold $\tau : |\chi^2 - \chi^2_{old}| \leq \tau$.

The edges weighing phase is again a data parallel application, where any arbitrary subset of the edges can be processed independently from the others by means of a parallel loop pattern. In order to fully exploit thread-level parallelism in multi-core computers, NuChart uses a skeletal approach [64] such as the *FastFlow* library [65, 66].

## 4. Computational background

The Grid paradigm represents a suitable solution to achieve the computational power required by many bioinformatic tools in a cost-effective way. However, most of the Grid environments present some challenges, i.e. to create a customized computing environment and its management, that can discourage some users in favour of Cloud solutions [3, 67]. But some technological solution can reduce the gap between the two paradigms, as the a Cloud-over-Grid approach [6, 68] and, more recently, the use of Docker in the Grid environment. In this work, we considered this last approach. In figure 2 the solution we designed is depicted, and details are presented in the following.

### *4.1. EGI Grid infrastructure*

EGI is a federation of computing and storage resource providers with the main goal to support research and development efforts. In particular EGI federates publicly funded data centers and Cloud providers, mostly in Europe, and give scientists access to more than 850,000 cores and 300 PetaBytes of disk storage. Its access is free for EGI members. As regards the Grid component, it relies basically on a geographically distributed model exploiting the Unified Middleware Distribution (UMD) [69], which offers to end-users a set of services that allow them to access and orchestrate all the available computing and storage resources.

The underlying architecture is composed by User Interfaces (UI), from which it is possible to submit jobs, described using the Job Description Language (JDL), to a Resource Broker (RB), a Grid Service able to find the most suitable computational resources for each job using different matching criteria [70]. Through the JDL file it is possible to specify a small file to be sent to the Grid resource, through the mechanism of the InputSandBox, a small file to retrieve, the so called OutputSandBox, and a number of global identifiers that refer to large files, previously allocated on the Grid distributed file system, that must be accessed during the computation. Moreover, the JDL file specifies the first command to launch on the grid, once the job is started.

Each computational site is accessed through a specific Grid services, the Computing Element (CE), which is an abstraction layer for the batch queue system installed on the site to handle the Worker Nodes (WNs), which are the actual resources where the computations are run. Concerning a Grid site, we can see the front-end CE as the master node of a cluster, and the WNs as the computing farm. On the other hand, each Grid site is completed by a Storage Element (SE), which stores the large files that are required for/produced by

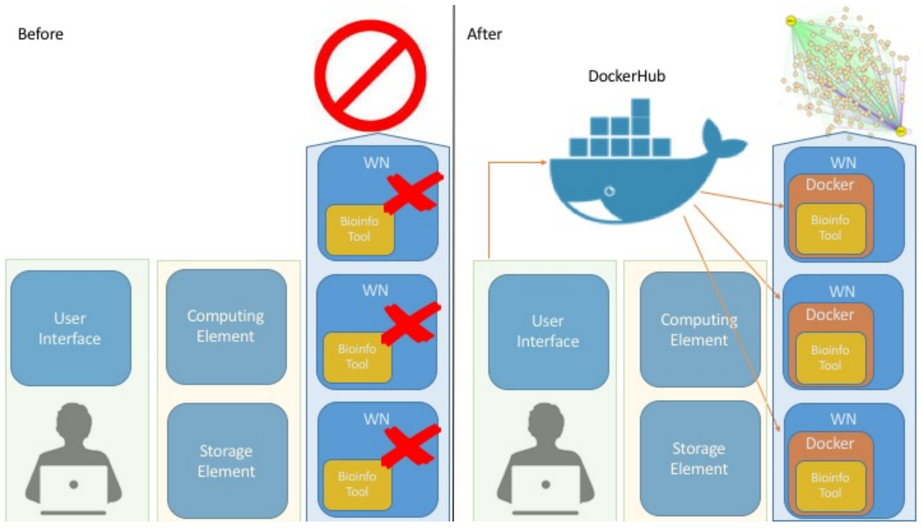

Figure 2: Schema of the proposed application. Due to the specific environment of the Working Nodes, the User is not able to run the bioinformatic application directly on Grid. On the other hand, using Docker and preparing a suitable Linux image for the application, the user is able to exploit the computational power of the Grid.

the computations. The network of the SEs, integrated by a File Metadata Service (FMS), implements a full distributed file systems, able to manage multiple replicas of each file.

Relying on this distributed infrastructure, the main steps performed by each job on the Grid sites are:

- download the executables from the InputSandBox and the input files from the closest available SE;
- run the simulation according to what is specified in the JDL file;
- send the output data back to the UI through the OutputSandBox.

On the UI, a job control facility monitors the status of the production, querying the proper Grid tools and orchestrating the productions triggering resubmission when needed. It also takes care of the data management of the final result. Security is demanded to the Grid Security Infrastructure (GSI), which relies on X.509 certificates for authentication on the computational and storage resources.

## *4.2. Udocker*

*Udocker* is a python2-based tool that can execute Docker containers in userspace, without requiring root privileges [71]. It enables basic download and execution of Docker containers by non-privileged users in Linux systems were the Docker server is not available. It can be used to access and execute the content of Docker containers in Linux batch and interactive clusters that are managed by other entities, such as Grid infrastructures. Udocker is an open source tool developed within the INDIGO-DataCloud European H2020 project [72] and has python2 as only prerequisite.

Udocker is a wrapper around several tools that mimic a subset of the Docker capabilities, including pulling images and running them with minimal functionality. The current implementation uses *PRoot* to emulate `chroot` without requiring privileges. This feature permits to use some tools that do not require root privileges, but still check the actual user id. For instance, software installation using `rpm`, `yum` or `dnf` inside the container is possible.

The containers data are unpacked and stored in the user home directory or other location of choice. Therefore, the containers data will be subject to the same filesystem protections as other files owned by the user. If containers have sensitive information, files and directories should be adequately protected by the user.

## 5. Experiment description

The analysis process performed by NuChart takes SAM files as input, which contain the sequences aligned against the reference genome. In the present tests we used anonymized human data from a public experiment [16], which do not present security issues. These files normally reach considerable sizes (order of GigaBytes), which are prohibitive for data movement across WNs through the InputSandBox. Therefore we exploited remote SEs to dispose a number of replicas for each file, as described in the following Section.

To perform a statistical analysis of the Grid performance in terms of input data size, we defined six classes, according to the sizes of the experiments we considered:

1. $SAM \leq 500MB$
2. $750MB < SAM \leq 1.0GB$
3. $1.5GB < SAM \leq 2.0GB$
4. $3.0GB < SAM \leq 4.0GB$
5. $6.0GB < SAM \leq 8.0GB$
6. $12.0GB < SAM \leq 16.0GB$

Output results are relatively small files that describe the chromosome conformation of the cell, basically a network in which nodes are genes and connections describe their proximity in the nucleus, and are sent back to the UI by using the OutputSandBox.

The application has been run on the Grid resources in a parallel fashion, using 4 cores and 16 GB of RAM, the same number of cores and RAM of our reference computer, which is equipped with a modern off-the-shelf Intel i5-7400 processor.

### *5.1. Set up of Storage Elements*

Considering that we used files having a size up to 16 GB we had to make use of the SE, because it is not possible for these files to travel with the job through the InputSandBox of the dispatching service. In order to avoid the overloading of remote SEs and to increase the availability of data files, we distributed them across multiple locations using the FMS functionalities. Through simple experiments (i.e hundreds of downloads of the same file) we evaluated the download efficiency and decided to set the replica multiplicity to a factor of 6. In fact, increasing the number of replicas above 6 had no significant effect on data management efficiency (greater than 90%), in terms of failed downloads (and hence failed jobs). About 10% of failures can be solved through multiple trials using different locations.

For what concerns the execution times, our experiment reported that, using 6 replicas, 90% of the downloads for 1 GB files took about 3 minutes, and we considered this number acceptable if compared with the scheduling and execution time of the application on the computational infrastructure, as discussed in Section 6. A time-out of 8 minutes was set on the job to consider the download failed and trigger a new trial using a different location. For the purposes of the work reported in this paper, we considered hundreds of concurrent downloads a reasonable scale. The jobs where instrumented to try as default SE the closest one to the site where they have been scheduled, and to randomly choose another replica in case of failure. Replicas are discovered dynamically by querying the Grid Logical File Catalogue (*gLite-LFC*).

### *5.2. Jobs set up*

The computation launched through the UI is actually executed on the WN and it is composed by the following sequence of steps:

1. Sanity checks of the environment, in particular the presence of the `/tmp` directory with proper permissions.

2. Download of the corresponding SAM file from the closest SE if available, otherwise choose another replica and download from there. Repeat until download completion – a fixed number of trials (10, to have a higher probability to try all 6 replicas) was set before labelling the download as failed. The SAM file is automatically mounted and available to the Docker image. Indeed, using Udocker, users can mount any host directory inside the container, although this is not a real mount but a `chroot`, and the directories will be visible inside the container.

3. Download the NuChart Docker image from *Docker Hub*. No multiple trials enabled. This is done, as usually for Docker, using different layers (in our case starting from a CentOS distribution).

4. Start the NuChart computation within the Udocker image and create the graph using all cores requested by the Grid job (set to 4 in our case).

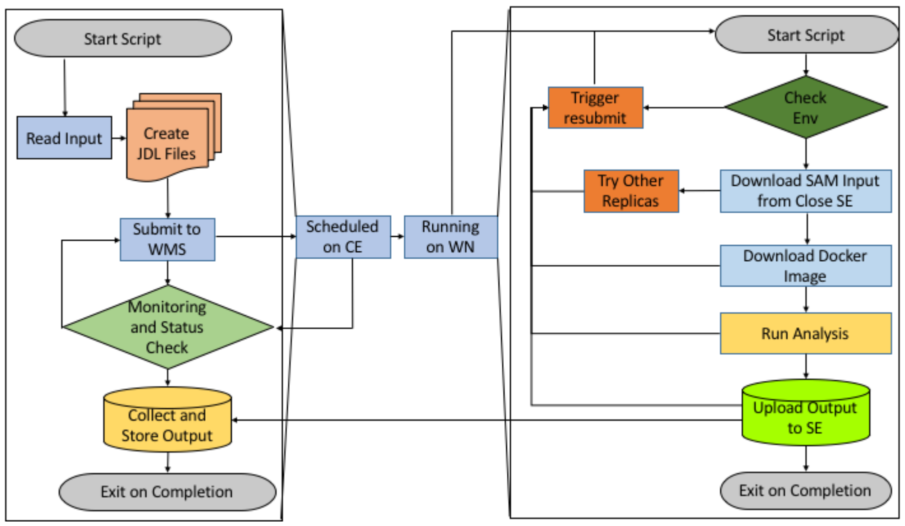

Figure 3: Jobs flowchart: job life cycle management framework running on the User Interface (on the left); once submitted jobs are routed to a suitable CE and eventually executed on a WN (center); main steps performed by the job on the WN (on the right).

5. Pack the output to reduce bandwidth usage during the upload.
6. Send back the output, report successful execution and exit.

In each step, entire job re-submission was triggered via standard Grid directives (JDL *Retry* and *ShallowRetry* options), in case of unrecoverable failures.

### *5.3. The NuChart Container and Experiment management*

We created a custom Docker image with an ad hoc installation of R equipped with all the add-on packages required to run NuChart, beside some mandatory system libraries such as the latest *boost* library, the GNU Scientific Library *gsl* and FastFlow. The necessity of using a custom R environment and some specific system libraries is the main reason why Grid cannot be exploited in a straightforward way for this application. Indeed, the installation of a custom R environment takes a lot of time, while the installation of system libraries requires root privileges.

The image is available at `https://hub.docker.com/r/imerelli/nuchart/` and, since it has been tagged as *latest*, it can be simply launched in Docker using the command `docker pull imerelli/nuchart`. Since Udocker is a very small application, it can be sent using the InputSandBox. On the other hand, the Udocker image is downloaded every time from the Docker repository. The only requirement is that the temporary directory (i.e. `/tmp`) of the server should be accessible and mounted without the *noexec* and *nosuid* flags, in order to make Udocker able to start.

As for any Grid computation, it is necessary to automate the creation of the JDL files that describe each job of the computational challenge. Then, the framework used to manage the computation launches each job, monitors their status, and finally works to collect the results. Due to the high number of jobs required, it is essential to develop the tools that perform all this process automatically, since it is not feasible to do it manually. This has been managed using a Python-based framework developed by us.

Figure 3 shows the flowchart of the jobs' life cycle management, whose steps consist in the creation of the JDL files, which results from the combination of all the input parameters, followed by the instructions for triggering all the jobs, monitoring their status and the retrieval of the results, when jobs are completed. For what concerns jobs distribution and monitoring, our submission framework exploits the gLite Workload Management System (WMS), while data management on the client side is based on the *lcg-utils* [73]. The framework needs to be executed on a standard gLite UI, while the initial data management (i.e., replica distribution of SAM files and Docker image uploads) have been completed before launching all the jobs.

## 6. Results and Discussion

Results achieved in this work are twofold. From one side, we tested the possibility of using Udocker to run on the EGI Grid infrastructure a bioinformatic application, describing the achievable performance and scalability figures. On the other hand, we analyzed the bioinformatic results, i.e. the contact maps describing the organization of the DNA in the nucleus for many different Hi-C experiments.

### *6.1. Udocker over Grid Performance*

Each experiment we performed consists in the submission of 100 jobs for each of the 6 input types, for a total of 600 jobs. Submitting 100 jobs to the WMS using a single thread on the UI took a negligible time (in the order of 2 minutes) with respect to the time needed to complete the full analysis, so as reference time for the performance evaluation we considered the time when all the jobs where dispatched to the WMS (i.e., *job_state = Submitted*). Other possible state for the jobs are: *Ready* (the job can be sent to the site batch system), *Scheduled* (the job is waiting on the batch system queue), *Running* (the job is executing), *Done* (the job completed) and *Aborted* (the job failed after all the re-submission trials). The Udocker version 1.1.1 has been used.

In order to compare the efficiency of Grid resources, in this work we performed two complete set of experiments, for a total of 1200 jobs. In the first simulation we restricted the number of sites eligible to run our jobs to a very limited set of trusted resources, named *selected resources*. They correspond to

2 CA, one in Padua[1] and the other in Naples[2]. They have been selected according to high job successful execution rate and low queue occupancy (in order to shorten waiting time as much as possible), and they have been very well tested before launching the jobs of the experiment. In the second simulation we used all the EGI sites accessible by our Virtual Organization *gridit*[3], named *free resources*. They correspond to 20 sites with 32 Computing Elements (i.e., different batch system queues) and 15 Storage Elements.

Moreover, we repeated the two complete set of experiments 5 times, in different days and time in order to increase the statistical power of our analysis, for a total of 6,000 jobs. In fact, using a production infrastructure, high variations can be observed due to the different level of usage of the communities having access to it.

Figure 4 shows the average queue time on the remote resources, the average input file download time, the average Docker setting time, and the average execution times of all the jobs for varying input size, considering both the *selected resources* and the *free resources*. Numerical data about lifetime cycle (average times and relative standard deviations) are reported Tables 1 and 2[4].

It can be observed that for small SAM files, the time that the jobs remain in the Scheduled state, and the time required to perform the data download is greater than the time required for the execution of the NuChart application. However, when increasing the size of the problem, the application requires more time, reaching the maximum for the largest SAM files. Considering only jobs that come to a successful completion, we have quite similar lifetime cycles. High standard deviations are explained by the fact that our tests were run in a production environment, where the incidental infrastructure load, due to community usage, is out of our control and heavily influences the progression of jobs execution.

Regarding the data download time, we notice that the average time remains quite similar for all input data, regardless of the size of the files to be downloaded. This seems to indicate that the influence of the number of jobs that download the same file is greater than the size of the file. The pull time of the Docker image from Docker Hub and the creation of the required container is relatively small in comparison with the application running time, becoming progressively negligible as the analyzed datasets increase in size.

Moreover, the running time of the algorithm depends on the number of connections identified in the SAM file. Although for larger files it is possible to forecast more connections, this is not always true. Therefore, it is difficult to estimate the execution time in each possible case. For instance, when the graph has few edges, the waiting time and download time can be greater than

[1]`http://operations-portal.egi.eu/vapor/resources/GL2ResSummaryServicesDetail?site=INFN-PADOVA&country=italy`

[2]`http://operations-portal.egi.eu/vapor/resources/GL2ResSummaryServicesDetail?site=GRISU-UNINA&country=italy`

[3] `https://operations-portal.egi.eu/vo/view/voname/gridit`

[4]Full results are available at https://github.com/imerelli/GridDocker

Table 1: Lifetime for selected resources, in minutes, waiting to be executed (WT), for downloading the data (DT), for the set up of the Docker execution environment (ST) and for the execution (ET))

| Class | Dimension | WT | DT | ST | ET |
|---|---|---|---|---|---|
| 1 | SAM $\leq$ 500MB | 24.7 $\pm$ 28.5 | 3.9 $\pm$ 1.0 | 5.0 $\pm$ 1.0 | 10.0 $\pm$ 1.0 |
| 2 | 750MB $<$ SAM $\leq$ 1GB | 24.2 $\pm$ 30.0 | 5.0 $\pm$ 1.0 | 5.0 $\pm$ 1.0 | 17.0 $\pm$ 1.0 |
| 3 | 1.5GB $<$ SAM $\leq$2.0 GB | 27.5 $\pm$ 32.1 | 5.1 $\pm$ 1.0 | 4.0 $\pm$ 1.1 | 24.9 $\pm$ 2.0 |
| 4 | 3.0GB $<$ SAM $\leq$4.0 GB | 23.5 $\pm$ 32.5 | 5.0 $\pm$ 1.0 | 5.0 $\pm$ 1.0 | 31.9 $\pm$ 2.0 |
| 5 | 6.0GB $<$ SAM $\leq$8.0 GB | 23.1 $\pm$ 34.9 | 8.0 $\pm$ 1.1 | 4.0 $\pm$ 1.0 | 32.7 $\pm$ 9.5 |
| 6 | 12.0GB $<$ SAM $\leq$16.0 GB | 22.9 $\pm$ 34.7 | 10.0 $\pm$ 1.0 | 4.0 $\pm$ 1.0 | 40.5 $\pm$ 10.6 |

Table 2: Lifetime for free resources, in minutes, waiting to be executed (WT), for downloading the data (DT), for the set up of the Docker execution environment (ST) and for the execution (ET))

| Class | Dimension | WT | DT | ST | ET |
|---|---|---|---|---|---|
| 1 | SAM $\leq$ 500MB | 41.5 $\pm$ 43.7 | 3.9 $\pm$ 1.0 | 5.1 $\pm$ 1.0 | 10.0 $\pm$ 1.0 |
| 2 | 750MB $<$ SAM $\leq$ 1GB | 39.1 $\pm$ 45.6 | 4.9 $\pm$ 1.0 | 5.0 $\pm$ 1.0 | 17.0 $\pm$ 1.0 |
| 3 | 1.5GB $<$ SAM $\leq$ 2GB | 40.5 $\pm$ 47.4 | 5.0 $\pm$ 1.1 | 4.0 $\pm$ 1.0 | 25.0 $\pm$ 2.0 |
| 4 | 3.0GB $<$ SAM $\leq$ 4GB | 40.8 $\pm$ 44.0 | 5.0 $\pm$ 1.0 | 4.0 $\pm$ 1.0 | 31.9 $\pm$ 2.1 |
| 5 | 6.0GB $<$ SAM $\leq$ 8.0GB | 44.5 $\pm$ 44.8 | 8.0 $\pm$ 1.0 | 5.0 $\pm$ 1.0 | 32.2 $\pm$ 9.6 |
| 6 | 12.0GB $<$ SAM $\leq$ 16.0GB | 44.2 $\pm$ 44.6 | 10.1 $\pm$ 1.0 | 5.0 $\pm$ 1.0 | 40.3 $\pm$ 10.7 |

the execution time. However, observing the time statistics we can conclude that the execution times corresponding to the application are generally much longer than the download time.

Figure 5 shows the number of completed jobs after the first job is completed, in the run that achieved median performances for each class of files. Even if relevant differences can be observed in runs on the same dataset executed in different times due to the infrastructure load, the time needed to analyze a single class of files (100 jobs for each class) varies from 100 $\pm$ 20 to 165 $\pm$ 20 minutes in the *selected resources* case, while is about 2.0/2.4 times greater in the *free resources* simulation, depending on the datasets. For this kind of application, and given the Grid resources load at the time of the test, the strategy that tries to maximize the job efficiency is also the one that minimizes the total duration of the production, and hence should be preferred.

Efficiency (overall percentage of jobs reported as successfully finished) and re-submissions (number of tries to achieve a successful job outcome) are two key parameters from a Grid computation and these are reported in Table 3, both for the *selected resources* and the *free resources.* In the *free resources* case we maximize the concurrency of the jobs (more jobs run in parallel) penalizing the job efficiency (i.e., more job re-submission after a failure are needed before having the job Done), while using the *selected resources* we maximize the job efficiency, but we have less concurrency, as reported in Table 3. In particular, we

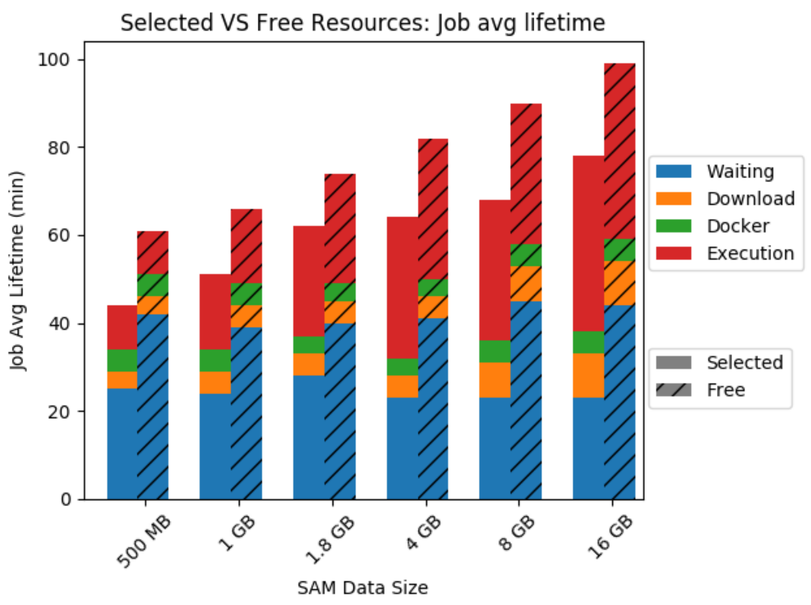

Figure 4: Average Jobs Lifetime, considering waiting time, input file download time, Docker setting time and running time. On the top panel the average job lifetime on the selected resource, on the bottom panel the average job lifetime on the whole Grid.

can see that we have a large number of failed jobs, due to the mis-configuration of some Grid sites, both in relation to the computational and storage resources, which causes errors in job submission and in data transfer respectively. The re-submissions number is particularly high in the case of the largest dataset (16GB) due to download timeout effects on the slower sites. This is in line with what has been reported so far [74, 75, 76]. On the other hand, basically all the computations are successful when jobs are dispatched to suitable CEs.

The crunching factor, which is defined (similarly to the speed up for parallel computations) as the expected single-CPU time required for the computations divided by the real computational time achieved on the distributed platform, is presented in Table 4, both for *selected resources* and *free resource*. Moreover, we reported the peak number of concurrent jobs (see Table 4), which is higher in the *free resources* because of its larger dimension compared to the *selected resources*, which nonetheless achieved a better scalability. Therefore these numbers show that the best result in terms of the execution time of the full experiment is likely to be achieved using a well tested - even if smaller infrastructure, due to the minimization of re-submissions.

The comparison with the results achieved by the previously published work [56], although relying on a different application, highlights three major points. First, in both the experiments, the pull time of the Docker image from Docker Hub and the creation of the required container is relatively small and basi-

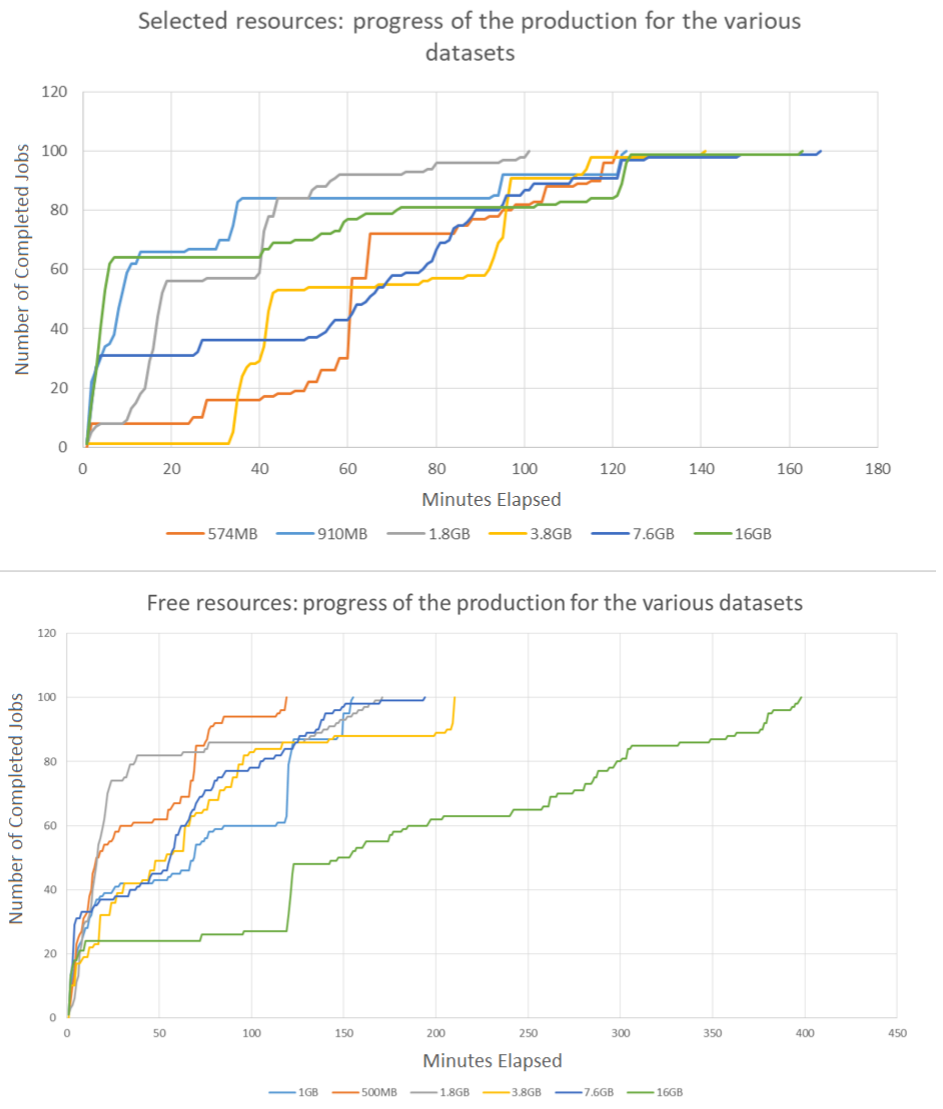

Figure 5: Progression of the challenge after the end of the first job for one of the performed run for each dataset. On the top panel the progression on the selected resources, on the bottom panel the progression on the whole Grid. This is shown for all input datasets considered in this work. We performed the same simulation five times for each dataset. High variations on the progression can be experienced depending on the infrastructure load. Details on average times and standard deviations for the job lifetime cycle are reported in the text and tables.

Table 3: Efficiency and Re-submissions

| | | Efficiency (%) | | Re-submissions (#) | |
|---|---|---|---|---|---|
| Class | Dimension | Selected | Free | Selected | Free |
| 1 | SAM $\leq$ 500MB | 100 | 90 $\pm$ 5 | 0 | 25 $\pm$ 3 |
| 2 | 750MB $<$ SAM $\leq$ 1GB | 100 | 78 $\pm$ 5 | 0 | 50 $\pm$ 15 |
| 3 | 1.5GB $<$ SAM $\leq$ 2GB | 100 | 86 $\pm$ 7 | 0 | 44 $\pm$ 10 |
| 4 | 3.0GB $<$ SAM $\leq$ 4GB | 100 | 92 $\pm$ 6 | 0 | 32 $\pm$ 10 |
| 5 | 6.0GB $<$ SAM $\leq$ 8.0GB | 100 | 75 $\pm$ 6 | 0 | 53 $\pm$ 7 |
| 6 | 12.0GB $<$ SAM $\leq$ 16.0GB | 100 | 60 $\pm$ 8 | 0 | 120 $\pm$ 15 |

Table 4: Crunching factor (CF) and peak number of concurrent jobs (PNJ). Average on multiple simulation performed in different times

| | | Average CF | | Average PNJ | |
|---|---|---|---|---|---|
| Class | Dimension | Selected | Free | Selected | Free |
| 1 | SAM $\leq$ 500MB | 7.6 $\pm$ 1.4 | 6.5 $\pm$ 1.3 | 35 $\pm$ 10 | 75 $\pm$ 5 |
| 2 | 750MB $<$ SAM $\leq$ 1GB | 17.2 $\pm$ 2.6 | 10.2 $\pm$ 2.0 | 30 $\pm$ 12 | 76 $\pm$ 6 |
| 3 | 1.5GB $<$ SAM $\leq$ 2GB | 24.3 $\pm$ 3.5 | 14.0 $\pm$ 2.7 | 31 $\pm$ 7 | 73 $\pm$ 4 |
| 4 | 3.0GB $<$ SAM $\leq$ 4GB | 26.2 $\pm$ 3.5 | 15.8 $\pm$ 2.7 | 32 $\pm$ 8 | 74 $\pm$ 5 |
| 5 | 6.0GB $<$ SAM $\leq$ 8.0GB | 19.2 $\pm$ 3.0 | 16.5 $\pm$ 3.0 | 40 $\pm$ 11 | 88 $\pm$ 8 |
| 6 | 12.0GB $<$ SAM $\leq$ 16.0GB | 18.4 $\pm$ 2.9 | 7.5 $\pm$ 2.6 | 35 $\pm$ 7 | 86 $\pm$ 6 |

cally negligible in comparison with the computational time. Second, it can be determined that, despite the generation of a delay due to the necessary data download and queuing time, this overhead is not exceedingly large, in particular while using selected resources, so it is still profitable to carry out the study in the Grid platform instead of being carried out on a local machine. In particular, considering our experiments, this claim is supported by the evaluation of the crunching factor, that shows a good scalability even for this small/medium data challenge. Third, in contrast with what has been previously reported, we did not experience an increase in the execution time. This was justified assuming that a Docker container is slower than the bare CE, but our data do not support this hypothesis.

## *6.2. Chromatin Conformation*

By using this Grid environment, we computed the graph-based representation for many public available Hi-C experiments. As an example, we show the results achieved analyzing data from the experiments of Lieberman-Aiden [16], which consist in four lines of karyotypically normal human lymphoblastoid cell line (GM06990) sequenced with Illumina Genome Analyzer, compared with two lines of K562 cells, an erythroleukemia human cell line with an aberrant karyotype.

The idea was to study the *Philadelphia translocation*, which is a specific chromosomal abnormality associated with chronic myelogenous leukemia (CML).

The presence of this translocation is a highly sensitive test for CML, since 95% of people with CML have this abnormality, although occasionally it may occur in acute myelogenous leukemia (AML). The result of this translocation is that a fusion gene created from the juxtaposition of the ABL1 gene on chromosome 9 (region $q34$) to part of the BCR (breakpoint cluster region) gene on chromosome 22 (region $q11$). This is a reciprocal translocation, creating an elongated chromosome 9 (called *der*9), and a truncated chromosome 22 (called the *Philadelphia* chromosome).

In this experiments, by using our graph-based approach, we compared the distance of some couples of genes that are known to create DNA translocations in CML/AML. The very interesting result is that ABL1 and BCR are likely to be distant 1 or 2 contacts ($p < 0.05$) in sequencing runs concerning GM06990, while they are directly in contact ($p < 0.05$) in sequencing runs related to K-562. Therefore, there is a perfect agreement between the positive and the negative presence of Hi-C interactions and cytogenetic data (see Figure 6).

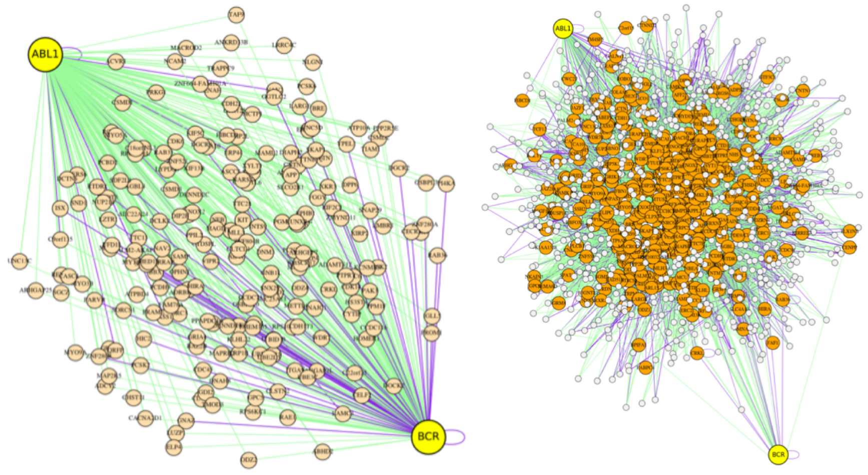

Figure 6: Example of graph computed during this analysis

## 7. Conclusions and Future Directions

This work presents the performance achieved by a compute-intensive bioinformatic study exploiting a new modelling approach for representing the chromosome conformation inside the nucleus using the Grid infrastructure provided by EGI and the container technology. Containers in fact extend the usability of Grid environments, since this technology allows users to make use of software that would not be executed without root permissions.

The key contribution is represented by the assessment of the feasibility of the use of Docker relying on an image which includes a previous installation of the used application and libraries in the Grid in terms of execution time and efforts to develop the container. The code of the application in fact has been adapted for its execution in this environment and the necessary tools have been developed for launching and monitoring the required jobs.

The achieved results show that the use of containers on the Grid, enabled by tools such as *Udocker*, can represent a fundamental advantage to make this paradigm accessible and suitable for many more users, that otherwise will rely on smaller, local clusters or possibly expensive Cloud infrastructures.

After assessing the feasibility of the Grid platforms with containers, the future directions of our work is represented by the development of an optimised version of the application using specific libraries that make a more efficient use of matrices. In this way, we would not have to perform a static compilation of the program to make it work in any CE: we could just compile it inside the Docker container, which is an even easier and powerful approach. Beside this, we plan to implement new containers for other bioinformatic applications and confirm the general validity of our findings.

## Acknowledgement

This work has been supported by the Italian Flagship InterOmics (grant number PB05) project, founded by the Italian Ministry of Education and Research, and the European MIMOMICS (grant number 305280) project, funded by the European Commissions 7th Framework Programme, The authors want to acknowledge the support of the INDIGO-DataCloud (grant number 653549) project, funded by the European Commissions Horizon 2020 Framework Programme. This work benefited from services provided to the GRIDIT Virtual Organisation by the national resource providers of the EGI Federation.

## References

[1] E.-G. Talbi, A. Y. Zomaya, Grid computing for bioinformatics and computational biology, Vol. 1, John Wiley & Sons, 2007.

[2] R. L. Rossi, R. M. Grifantini, Big data: challenge and opportunity for translational and industrial research, Frontiers in Digital Humanities 5 (2018) 13.

[3] A. J. Banegas-Luna, B. Imbernón, A. Llanes Castro, A. Perez-Garrido, J. P. Ceron-Carrasco, S. Gesing, I. Merelli, D. DAgostino, H. Perez-Sanchez, Advances in distributed computing with modern drug discovery, Expert opinion on drug discovery 14 (1) (2019) 9–22.

[4] E. Foundation, Egi use cases (2017).
URL `https://zenodo.org/record/159455#.W8MIeC9aZvU`

[5] I. Foster, C. Kesselman, The history of the grid, Advances in Parallel Computing 20 (2011) 3–30. `doi:10.3233/978-1-60750-803-8-3`.

[6] I. Merelli, P. Cozzi, E. Ronchieri, D. Cesini, D. DAgostino, Porting bioinformatics applications from grid to cloud: A macromolecular surface analysis application case study, The International Journal of High Performance Computing Applications 31 (3) (2017) 182–195.

[7] D. Merkel, Docker: Lightweight Linux containers for consistent development and deployment, Linux J. 2014 (239).

[8] A. Silver, Software simplified, Nature News 546 (7656) (2017) 173.

[9] E. de Wit, W. de Laat, A decade of 3C technologies: insights into nuclear organization, Genes & Development 26 (1) (2012) 11–24.

[10] S. Sati, G. Cavalli, Chromosome conformation capture technologies and their impact in understanding genome function, Chromosoma 126 (1) (2017) 33–44. `doi:10.1007/s00412-016-0593-6`.
URL `https://doi.org/10.1007/s00412-016-0593-6`

[11] L. Harewood, K. Kishore, M. D. Eldridge, S. Wingett, D. Pearson, S. Schoenfelder, V. P. Collins, P. Fraser, Hi-c as a tool for precise detection and characterisation of chromosomal rearrangements and copy number variation in human tumours, Genome biology 18 (1) (2017) 125.

[12] K. P. Eagen, Principles of chromosome architecture revealed by hi-c, Trends in biochemical sciences.

[13] J. Fraser, I. Williamson, W. A. Bickmore, J. Dostie, An overview of genome organization and how we got there: from fish to hi-c, Microbiology and Molecular Biology Reviews 79 (3) (2015) 347–372.

[14] Y. Shavit, I. Merelli, L. Milanesi, P. Lio, How computer science can help in understanding the 3d genome architecture, Briefings in bioinformatics 17 (5) (2015) 733–744.

[15] J. Dekker, K. Rippe, M. Dekker, N. Kleckner, Capturing Chromosome Conformation, Science 295 (5558) (2002) 1306–1311.

[16] 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, J. Dekker, Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome, Science 326 (5950) (2009) 289–293.

[17] I. Merelli, P. Liò, L. Milanesi, Nuchart: An r package to study gene spatial neighbourhoods with multi-omics annotations, PLoS ONE 8 (9) (2013) e75146.

[18] F. Tordini, M. Drocco, I. Merelli, L. Milanesi, P. Liò, M. Aldinucci, NuChart-II: a graph-based approach for the analysis and interpretation of Hi-C data, in: Post-Conference proceedings of the 11th Intl. meeting on Computational Intelligence methods for Bioinformatics and Biostatistics (CIBB 2014), Vol. 8623 of LNBI, Springer, Cambridge, UK, 2015, to appear.

[19] F. Tordini, I. Merelli, P. Liò, L. Milanesi, M. Aldinucci, NuchaRt: embedding high-level parallel computing in R for augmented Hi-C data analysis, in: S. I. Publishing (Ed.), Computational Intelligence Methods for Bioinformatics and Biostatistics, Vol. 9874 of Lecture Notes in Computer Science, Springer International Publishing, Cham (ZG), 2016, pp. 259–272.

[20] F. Tordini, M. Drocco, C. Misale, L. Milanesi, P. Liò, I. Merelli, 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, 2015.

[21] N. Kratzke, Lightweight virtualization cluster how to overcome cloud vendor lock-in, Journal of Computer and Communications 2 (12) (2014) 1.

[22] P.-C. Quint, N. Kratzke, Overcome vendor lock-in by integrating already available container technologies towards transferability in cloud computing for smes, CLOUD COMPUTING 2016 (2016) 50.

[23] J. T. Dudley, A. J. Butte, In silico research in the era of cloud computing, Nature biotechnology 28 (11) (2010) 1181.

[24] B. Howe, Virtual appliances, cloud computing, and reproducible research, Computing in Science & Engineering 14 (4) (2012) 36–41.

[25] D. Marshall, Understanding full virtualization, paravirtualization, and hardware assist, VMWare White Paper (2007) 17.

[26] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: ACM SIGOPS operating systems review, Vol. 37, ACM, 2003, pp. 164–177.

[27] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, kvm: the linux virtual machine monitor, in: Proceedings of the Linux symposium, Vol. 1, Dttawa, Dntorio, Canada, 2007, pp. 225–230.

[28] M. Rosenblum, Vmwares virtual platform, in: Proceedings of hot chips, Vol. 1999, 1999, pp. 185–196.

[29] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, L. Peterson, Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors, in: ACM SIGOPS Operating Systems Review, Vol. 41, ACM, 2007, pp. 275–287.

[30] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An updated performance comparison of virtual machines and linux containers, in: Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On, IEEE, 2015, pp. 171–172.

[31] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, W. Zhou, A comparative study of containers and virtual machines in big data environment, arXiv preprint arXiv:1807.01842.

[32] C. Anderson, Docker [software engineering], IEEE Software 32 (3) (2015) 102–c3. `doi:10.1109/MS.2015.62`.
URL `doi.ieeecomputersociety.org/10.1109/MS.2015.62`

[33] V. G. da Silva, M. Kirikova, G. Alksnis, Containers for virtualization: An overview, Applied Computer Systems 23 (1) (2018) 21–27.

[34] Y. Zheng, D. M. Nicol, A virtual time system for openvz-based network emulations, in: Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, 2011, pp. 1–10.

[35] K. Hightower, B. Burns, J. Beda, Kubernetes: Up and Running: Dive Into the Future of Infrastructure, " O'Reilly Media, Inc.", 2017.

[36] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, I. Stoica, Mesos: A platform for fine-grained resource sharing in the data center., in: NSDI, Vol. 11, 2011, pp. 22–22.

[37] R. Peinl, F. Holzschuher, F. Pfitzer, Docker cluster management for the cloud-survey results and own solution, Journal of Grid Computing 14 (2) (2016) 265–282.

[38] Amazon, Aws documentation.
URL `https://docs.aws.amazon.com/ecs/index.html#lang/en_us`

[39] Microsoft, Azure container service documentation.
URL `https://docs.microsoft.com/en-us/azure/container-service/`

[40] Rancher documentation.
URL `https://rancher.com/docs/rancher/v1.6/en/`

[41] S. V. Angiuoli, M. Matalka, A. Gussman, K. Galens, M. Vangala, D. R. Riley, C. Arze, J. R. White, O. White, W. F. Fricke, Clovr: a virtual machine for automated and portable sequence analysis from the desktop using cloud computing, BMC bioinformatics 12 (1) (2011) 356.

[42] E. Afgan, B. Chapman, M. Jadan, V. Franke, J. Taylor, Using cloud computing infrastructure with cloudbiolinux, cloudman, and galaxy, Current protocols in bioinformatics 38 (1) (2012) 11–9.

[43] C. Boettiger, An introduction to docker for reproducible research, ACM SIGOPS Operating Systems Review 49 (1) (2015) 71–79.

[44] L.-H. Hung, D. Kristiyanto, S. B. Lee, K. Y. Yeung, Guidock: using docker containers with a common graphics user interface to address the reproducibility of research, PloS one 11 (4) (2016) e0152686.

[45] F. Moreews, O. Sallou, H. Ménager, et al., Bioshadock: a community driven bioinformatics shared docker-based tools registry, F1000Research 4.

[46] E. Afgan, D. Baker, B. Batut, M. Van Den Beek, D. Bouvier, M. Čech, J. Chilton, D. Clements, N. Coraor, B. A. Grüning, et al., The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update, Nucleic acids research 46 (W1) (2018) W537–W544.

[47] F. Moreews, O. Sallou, H. Ménager, et al., Bioshadock: a community driven bioinformatics shared docker-based tools registry, F1000Research 4.

[48] C. Jansen, M. Witt, D. Krefting, Employing docker swarm on openstack for biomedical analysis, in: International Conference on Computational Science and Its Applications, Springer, 2016, pp. 303–318.

[49] A. A. Ali, M. El-Kalioby, M. Abouelhoda, The case for docker in multi-cloud enabled bioinformatics applications, in: International Conference on Bioinformatics and Biomedical Engineering, Springer, 2016, pp. 587–601.

[50] M. Concas, D. Berzano, S. Bagnasco, S. Lusso, M. Masera, M. Puccio, S. Vallero, Plancton: an opportunistic distributed computing project based on docker containers, in: Journal of Physics: Conference Series, Vol. 898, IOP Publishing, 2017, p. 092049.

[51] S. Smith, Containerizing the grid - boinc on docker (2015).
URL `https://rsmitty.github.io/Containerizing-The-Grid/`

[52] B. I. Ismail, E. M. Goortani, M. B. Ab Karim, W. M. Tat, S. Setapa, J. Y. Luke, O. H. Hoe, Evaluation of docker as edge computing platform, in: Open Systems (ICOS), 2015 IEEE Confernece on, IEEE, 2015, pp. 130–135.

[53] T. P. Morgan, Bridging the gap between grid and containers (2015).
URL `https://www.nextplatform.com/2015/12/04/bridging-the-gap-between-grid-and-containers/`

[54] M. Gholap, Deploying selenium grid using docker (2018).
URL `https://dzone.com/articles/deploying-selenium-grid-using-docker`

[55] N. Medjkoune, Integrating docker containers into the cern batch system (2016).
URL `https://zenodo.org/record/159455#.W8MIeC9aZvU`

[56] M. Chillarón, V. Vidal, D. Segrelles, I. Blanquer, G. Verdú, Combining grid computing and docker containers for the study and parametrization of ct image reconstruction methods, Procedia Computer Science 108 (2017) 1195–1204.

[57] I. Chepelev, G. Wei, D. Wangsa, Q. Tang, K. Zhao, Characterization of genome-wide enhancer-promoter interactions reveals co-expression of interacting genes and modes of higher order chromatin organization, Cell Res 22 (3) (2012) 490–503.

[58] F. Tordini, M. Aldinucci, L. Milanesi, P. Liò, I. Merelli, The genome conformation as an integrator of multi-omic data: The example of damage spreading in cancer, Frontiers in Genetics 7 (194) (2016) 1–17.

[59] I. Merelli, F. Tordini, M. Drocco, M. Aldinucci, P. Liò, L. Milanesi, Integrating multi-omic features exploiting Chromosome Conformation Capture data, Frontiers in Genetics 6 (40).

[60] S. Wingett, P. Ewels, M. Furlan-Magaril, T. Nagano, S. Schoenfelder, P. Fraser, S. Andrews, HiCUP: pipeline for mapping and processing Hi-C data [version 1; referees: 2 approved, 1 approved with reservations], F1000Research 4 (1310).

[61] S. Hong, T. Oguntebi, K. Olukotun, Efficient parallel graph exploration on multi-core CPU and GPU, in: Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques, PACT '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 78–88.

[62] E. Yaffe, A. Tanay, Probabilistic modeling of hi-c contact maps eliminates systematic biases to characterize global chromosomal architecture, Nature Genetics 43 (11) (2011) 1059–1065.

[63] J. A. Nelder, R. W. M. Wedderburn, Generalized linear models, Journal of the Royal Statistical Society, Series A, General 135 (1972) 370–384.

[64] M. Aldinucci, S. Campa, M. Danelutto, P. Dazzi, D. Laforenza, N. Tonellotto, P. Kilpatrick, Behavioural skeletons for component autonomic management on grids, in: Making Grids Work, Springer, 2008, pp. 3–15.

[65] M. Aldinucci, M. Danelutto, P. Kilpatrick, M. Torquati, Fastflow: high-level and efficient streaming on multi-core, in: S. Pllana, F. Xhafa (Eds.), Programming Multi-core and Many-core Computing Systems, Parallel and Distributed Computing, Wiley, 2017, Ch. 13, pp. 261–280.

[66] M. Danelutto, M. Torquati, Loop parallelism: a new skeleton perspective on data parallel patterns, in: Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on, IEEE, 2014, pp. 52–59.

[67] D. D'Agostino, A. Clematis, A. Quarati, D. Cesini, F. Chiappori, L. Milanesi, I. Merelli, Cloud infrastructures for in silico drug discovery: Economic and practical aspects, BioMed Research International 2013.

[68] E. Ronchieri, D. Cesini, D. D'Agostino, V. Ciaschini, G. Dalla Torre, P. Cozzi, D. Salomoni, A. Clematis, L. Milanesi, I. Merelli, The wnodes cloud virtualization framework: a macromolecular surface analysis application case study, in: Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on, IEEE, 2014, pp. 218–222.

[69] EGI, Egi software repository.
URL `http://repository.egi.eu`

[70] A. Clematis, A. Corana, D. DAgostino, A. Galizia, A. Quarati, Job–resource matchmaking on grid through two-level benchmarking, Future Generation Computer Systems 26 (8) (2010) 1165–1179.

[71] J. Gomes, E. Bagnaschi, I. Campos, M. David, L. Alves, J. Martins, J. Pina, A. Lopez-Garcia, P. Orviz, Enabling rootless linux containers in multi-user environments: the udocker tool, Computer Physics Communications 232 (2018) 84–97.

[72] D. Salomoni, I. Campos, L. Gaido, G. Donvito, M. Antonacci, P. Fuhrman, J. Marco, A. Lopez-Garcia, P. Orviz, I. Blanquer, et al., Indigo-datacloud: foundations and architectural description of a platform as a service oriented to scientific computing, CoRR abs/1711.01981. `arXiv:1711.01981`.
URL `http://arxiv.org/abs/1711.01981`

[73] B. Stephen, C. Simone, L. Elisa, L. Maarten, M. L. Patricia, M. Vincenzo, N. Christopher, S. Roberto, S. Andrea, gLite 3.2 user guide, `https://edms.cern.ch/file/722398/1.4/gLite-3-UserGuide.pdf`, version 1.4.2, July 18, 2012.

[74] H.-C. Lee, J. Salzemann, N. Jacq, H.-Y. Chen, L.-Y. Ho, I. Merelli, L. Milanesi, V. Breton, S. C. Lin, Y.-T. Wu, Grid-enabled high-throughput in silico screening against influenza a neuraminidase, IEEE transactions on nanobioscience 5 (4) (2006) 288–295.

[75] I. Merelli, D. Pescini, E. Mosca, P. Cazzaniga, C. Maj, G. Mauri, L. Milanesi, Grid computing for sensitivity analysis of stochastic biological models, in: International Conference on Parallel Computing Technologies, Springer, 2011, pp. 62–73.

[76] G. Degliesposti, V. Kasam, A. Da Costa, H.-K. Kang, N. Kim, D.-W. Kim, V. Breton, D. Kim, G. Rastelli, Design and discovery of plasmepsin ii inhibitors using an automated workflow on large-scale grids, ChemMedChem: Chemistry Enabling Drug Discovery 4 (7) (2009) 1164–1173.