# A storage policy for a hybrid federated cloud platform executing bioinformatics applications

Deric Lima, Breno Moura, Gabriel Oliveira, Edward Ribeiro, Aletéia Araújo, Maristela Holanda, Roberto Togawa and Maria Emilia Walter

**Abstract**—Bioinformatics tools require large-scale processing mainly due to very large databases achieving gigabytes of size. In federated cloud environments, although services and resources may be shared, storage is particularly difficult, due to distinct computational capabilities and data management policies of several separated clouds. In this work, we propose a storage policy for BioNimbuZ, a hybrid federated cloud platform designed to execute bioinformatics applications. Our storage policy, BioClouZ, aims to perform efficient choices to distribute and replicate files to the best available cloud resources in the federation in order to reduce computational time. BioClouZ uses four parameters - latency, uptime, free size and cost, weighted (according to *ad hoc* tests) to model their influences to data storage and recovery. Experiments were performed with real biological data executing a commonly used tool to map short reads in a reference genome in BioNimbuZ, composed of clouds executing in Amazon EC2, Azure and University of Brasilia. The results showed that, when compared to the random algorithm first used in BioNimbuZ, the BioClouZ policy significantly improved the total execution time due to more efficient choices of the clouds to store the files. Other bioinformatics applications can be used with BioClouZ in BioNimbuZ as well, since the platform was designed independently from particular tools and databases.

**Keywords**—storage policy, hybrid federated cloud platform, bioinformatics.

## 1 Introduction

Advances in genome sequencing techniques allowed to generate enormous volumes of genomic sequences. These sequences are stored in files (databases) that have to be analyzed using bioinformatics tools requiring significant computational resources. In this scenario, cloud computing can improve execution of bioinformatics applications in an efficient and collaborative way. Many specific bioinformatics applications executing in single cloud environments have been presented in the literature [1], [2], [3], [4], [5], [6], [7].

In 2012, our group proposed BioNimbus [8], a hybrid federated cloud architecture to execute bioinformatics workflows. BioNimbus was designed to provide functionalities that allow to join clouds belonging to different organizations and having different policies, shared in a consistent and controlled way. A first prototype was constructed, and one workflow using real genomic data was executed in the platform. The experiments showed that the time to execute this workflow in BioNimbus was improved when compared to its execution in one powerful machine.

BioNimbus platform was implemented using a simple random storage policy, the first cloud presenting enough space was chosen to store files. In the tests, this policy frequently generated a time overhead, since storing one file $F$ in a cloud not chosen by the scheduler to execute a task $T$ using $F$ implied in a second transfer of $F$ to the cloud actually executing $T$. In other words, a cloud executing one task using files stored in another cloud implied in extra time to transfer these input files, before actually start running. This strongly affected the total time, since files containing genomic data usually have gigabytes of size.

In this context, this work aims to propose a novel storage policy to BioNimbus, in order to

- *D. Lima, B. Moura, G. Oliveira, E. Ribeiro, A. Araújo, M. Holanda and M. E. Walter are with the Department of Computer Science, University of Brasilia, Brasilia, Brazil. E-mails: {dericlima, mourabreno, gabriel_222zt, edward.ribeiro}@gmail.com, {aleteia, mholanda, mia}@cic.unb.br*
- *R. Togawa is with the Bioinformatics Laboratory of EMBRAPA/ Genetic Resources and Biotechnology, Brasilia, Brasil. E-mail: roberto.togawa@embrapa.br.*

*Manuscript sent February 28, 2014*

perform an efficient choice to distribute files and reduce the total execution time, considering free storage space and charge of each cloud as well as network latency and time that a machine remains working (called *uptime*) in the federation. Therefore, the proposed storage policy, called BioClouZ, is based on four parameters - free space, cost, latency and uptime, weighted according to their influences to data storage and recovery. *Ad-hoc* tests and results from the literature were used to define these weights.

To implement BioClouZ, a coordination service was included in BioNimbus, and the communication service among the federated clouds was changed. BioNimbus architecture was completely reimplemented, which led to a second version, called BioNimbuZ, a more scalable, dynamic and easy to program platform, using Zookeeper [9] and Avro [10].

## 2 BioNimbuZ platform

In this section, we first describe the BioNimbus architecture [8]. After, we show how Zookeeper [9] and Avro [10] were used to generate the BioNimbuZ platform.

### 2.1 BioNimbus architecture

The BioNimbus architecture [8] is structured in three layers: application, core and infrastructure (see Figure 1). All the components and respective functions are clearly defined and separated, allowing simple and efficient inclusion of new clouds, transparently to users.

The *infrastructure layer* includes *plugins*, used to map the communication among the federated clouds and the *core layer* management services, which offer computational resources, e.g., virtual machines, data storage and networks. The *application layer* is the communication interface between BioNimbus and users.

The *core layer* was designed to provide services that integrate the clouds belonging to the federation. The *job controller* connects the *core* and the *application layers*, being responsible for managing the requests of one user and the current status of the execution of these requests, which can be consulted at any time. It also

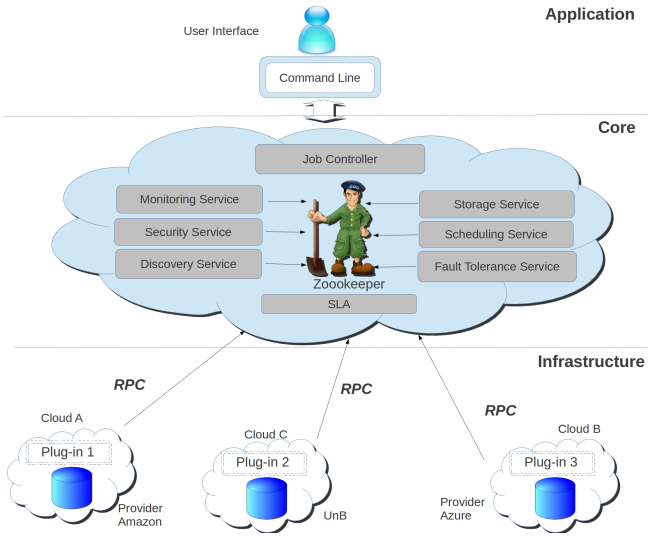


Fig. 1. BioNimbus federated cloud architecture.

controls users' access rights, using the security service to verify his/her credentials.

The *monitoring service* manages the status of each cloud, i.e., it notifies all the federated clouds when a cloud enters or leaves the federation. This service is also responsible to warrant the execution of the requested tasks.

The *security service* guarantees integration among the clouds in a secure way, through access rules, which have to be carried out according to the policies and requirements of each federated cloud. It also assures a secure communication among users and service providers, and offers access control policies for a cloud entering in the federation.

The *discovery service* finds the clouds integrating the federation, and consolidates information about their storage, processing and transfer capabilities, and also the offered bioinformatics tools, together with details of execution parameters and input/output files.

The *storage service* manages the files used in the bioinformatics applications, as well as recovers and controls data transferring.

The *scheduling service* coordinates task execution, using efficient scheduling policies to distribute these tasks among the federated clouds.

The *fault tolerance service* guarantees that all the control services in the *core layer* are available. For this, it registers each of the managing services, monitoring them with messagens sent in short intervals. When a service does not

answer after a fixed time, the fault tolerance service initiates an election algorithm to define which server has to initiate another instance of the failed service. When this process finishes, all the clouds are warned about the new choice.

A Service-Level Agreement (SLA) controler is a formal contract between service providers and clients to guarantee *Quality-of-Service* (QoS) to the users [11]. In BioNimbus, the *SLA controler* is responsible for implementing the SLA life cycle, in six steps: service provider discovery; SLA definition; agreement establishment; agreement monitoring violation; contract finishing; and imposing of penalties due to agreement violation. To identify agreement parameters, SLA templates are used, representing, among others, QoS parameters negotiated between the user and the BioNimbus architecture.

### 2.2 BioNimbuZ: a new implementation

Now, we show the way Zookeper and Avro were used to create the BioNimbuZ platform.

#### *ZooKeeper*

Building distributed services is often complex and error-prone, e.g., controlling conditions of executions and treating deadlocks. Apache ZooKeeper [9] is an open source service that facilitates to develop distributed applications, being responsible for the implementation of coordination services. It provides a simple set of primitives designed to implement high level services. e.g., synchronization, maintenance, configuration and group processing.

ZooKeeper store, among others, data coordination, status information, setting and location information, using *znodes*, data structures similar to directories. An Access Control List (ACL) is associated to each *znode*, restricting who can read/write it. ZooKeeper has also the concept of *ephemeral znode*, which exists while the session creating it is active, being deleted when this session is not active any more. Each session corresponds to a connected client.

Another important concept in ZooKeeper is the *watcher*, a notice event. A *watcher* is triggered and removed when changes occur in the corresponding *znode*. When a *watcher* is triggered, the client receives a message noticing that its *znode* has been changed. If the connection between the client and one ZooKeeper server is interrupted, the client receives a message stating that the server has crashed. *Watchers* and *znodes* can be defined by each server.

#### *Avro*

Apache Avro [10] is a system of data serialization and RPC (Remote Procedure Call). The main features of this system are: rich data structure described by schemes; binary data in a compact and fast format; a file container to store persistent data; use of RPC; and simple integration with dynamic languages, i.e., it is not necessary to code for reading and writing files nor to use and implement RPC protocols.

Avro data is stored in a file together with its scheme, being Avro data read together with its corresponding scheme. This makes data serialization efficient and easy, since data and corresponding schemes are self-explanatory, facilitating the use of dynamic script languages.

In RPC, client and server exchange data schemes to make a connection. Since client and server have the schemes one of the other, correspondance between the same fields are easily solved. Avro schemes are defined with JSON [12], which facilitates the implementation in languages also presenting JSON libraries.

#### *BioNimbuZ platform*

Using Avro and Zookeper, we modified the first prototype of the BioNimbus architecture proposed in Saldanha *et al.* [8], in order to have a more scalable, dynamic, and easy to manage platform, called BioNimbuZ. In particular, its communication and coordination mechanisms were completely modified.

The communication mechanism in the first BioNimbus platform was implemented through P2P messaging [13], being its control distributed among each federated cloud. The discovery service (for clouds belonging to the federation and their respective resources) was also implemented through this P2P network. In contrast, in the new version, communication among the federated clouds has been replaced by RPC Avro [10]. While with P2P, a broadcast was needed (e.g., to find out the clouds integrating the federation), with RPC, a message

is sent directly to the destination cloud. Thus, Avro allowed to provide a simpler and more robust communication interface.

Besides, when a cloud enters in the federation, a *znode* is created with its information, and consulting or updating this cloud information can be easily done with ZooKeeper [9]. This allows to control the federated clouds in a decentralized way. Thus, coordination in BioNimbuZ was implemented using *znodes* (see Figure 2), which contain the service modules.

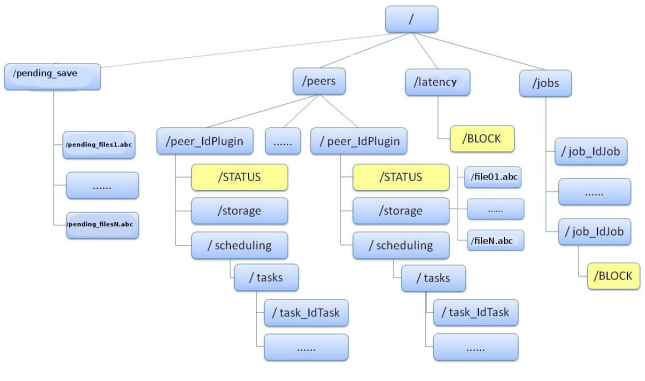


Fig. 2. Structure of the *znodes* in BioNimbuZ.

Therefore, when a new cloud $C$ enters in the federation, one *STATUS znode* is created to store its available resources. This *ephemeral znode* exists until $C$ loses its connection, which indicates to Zookeeper that $C$ left the federation. A cloud running a BioNimbuZ instance, registered in a *znode*, is called **BioNimbuZ server**.

## 3 BioClouZ storage policy

The first prototype of the BioNimbus architecture [8] had a very simple random storage policy, as said before. Our novel BioClouZ policy use information from one ZooKeeper *znode* to compute a storage value for each cloud.

In the experiments developed with the first BioNimbus prototype, we noted that a bioinformatics application ran efficiently if the input files were already stored in the cloud chosen to execute it [8]. Therefore, a policy warranting that the input files of one application are stored in the cloud chosen to perform the task, or near it, would certainly improve performance. We intended to model this in the BioClouZ policy.

BioClouZ controls how data is stored, deleted and replicated in BioNimbuZ. To control stored data, BioClouZ uses the ZooKeeper *znodes*. Each BioNimbuZ server entering in the federation receives an unique identifier (ID), being a *znode* created with this ID, together with the *STATUS znode*, which sends to the monitoring service the resource information of this BioNimbuZ server. When a BioNimbuZ server leaves the federation, the *STATUS znode* is changed to *STATUS WAITING*, which asks the monitoring service to immediately initiate the fault tolerance service for this server.

A set of children *znodes* models the files belonging to one BioNimbuZ server. For each file, a *znode* is created with its information (ID, name, size and a *pluginlist*). The *pluginlist* is a list of all the BioNimbuZ servers storing this file, since the storage service automatically replicates all data stored in the federation. This process is detailed in Section 3.2.

Besides, a *pending_save znode* is created with the information of each file stored in the federation, being all the BioNimbuZ servers noticed about these files (see Figure 2). This *pending_save znode* is deleted only when the file is properly stored in one BioNimbuZ server.

The idea of the BioClouZ storage policy is to compute a storage value for each BioNimbuZ server and, based on these values, choose the servers with the lower values to store and replicate the files to be stored in BioNimbuZ. This computation is shown next.

### 3.1 Storage value

A storage value is computed for each file saved or replicated in the BioNimbuZ federation. This computation uses information between the client who requested the task and each BioNimbuZ server with enough disk space to store the file, transparently to the user. Three variables are used to compute the storage value for one BioNimbuZ server:

- Free space size ($F$): free disk space;
- Uptime ($U$): time the BioNimbuZ server remains working in the federation, i.e., a server working for a longer time appears to be more reliable to store the file, when compared to others presenting

lower times (due to leaving or recently integrating the federation);
- Latency ($L$): latency between the client and the BioNimbuZ server, since it is important to know if the file transfer to the destination will be fast, or not.

These variables influence the storage value in different ways, thus uptime was more heavily weighted than the free space size. Regarding uptime, whenever a BioNimbuZ server leaves the federation, the value of its uptime is reset, and counting restarts if it re-enters in the federation. Resetting the uptime does not affect the computation of the storage values of stable servers since, as shown by Maymounkov and Maziéres [14] and Verespej and Pasquale [15], stable servers "remains stable" even if the uptime is reset (due to some failure). Or, if one server remains working for one hour tends to continue working for more one hour, which is different from servers that usually fail.

Our storage value was based on Xun-Yi and Xiao-Dong [16], who proposed that the storage value of one file $F$ could be computed from the cost of transferring $F$ to the destination added to the cost of storing each byte of $F$.

BioClouZ computes a storage value for each BioNimbuZ server relative to each client. When one client gets a list of the BioNimbuZ servers in order to calculate their storage values, the computation is done so that one storage value can not be overwritten by another one, if a second client makes a request before the client who had requested first receives his storage values. Besides, the storage value computed by BioClouZ models the transfer time between two servers, the lower value indicating the lower cost to transfer the file.

Equation 1 shows the computation of the storage value ($S$), where the three variables $U$ (uptime), $F$ (free space time) and $L$ (latency) were weighted by $\gamma$, $\beta$ and $\alpha$, respectively, such that $\alpha+\beta+\gamma = 1.0$. Based on several *ad hoc* tests, the weights were assigned to the following values: $\alpha = 0.5$; $\beta = 0.2$; and $\gamma = 0.3$. Moreover, since BioNimbuZ is a hybrid platform, public or private clouds can integrate it. Thus it is possible that a BioNimbuZ server charges for storage, adding an extra cost to the storage value. Some clouds, e.g. Amazon and Azure, charges for gigabytes of stored data. This cost, dollars per gigabyte, is included in the configuration file of the corresponding BioNimbuZ server, and this storage price was modeled by the variable *Cost*. In clouds not charging for storage, this variable is assigned to 0.

$$S = 1/(U * \gamma + F * \beta) * (L * \alpha) + Cost \qquad (1)$$

After computing the storage value for each BioNimbuZ server, a client receives a list containing all the server storage values, ordered in ascending order, i.e., from the server with the lowest cost to the one with the largest. Next, this list is returned to the client who requested the upload, and BioClouZ tries to send the file to the server appearing first in this list. If it is not possible, another trial is made to the next server, and this continues until the file is stored.

Figure 3 shows, in a simplified form, how BioNimbuZ uses Avro for a file upload: (1) the client makes a connection to the federation; (2) the client receives a list of BioNimbuZ servers that may receive the file; (3) the client computes latency between himelf and each BioNimbuZ server, sending one RPC call to the servers that may store the file (these are the latency values used by BioClouZ). Next, the best BioNimbuZ server to store the file will be found based on the values computed by BioClouZ; (4) the first BioNimbuZ server makes one RPC call to the server chosen by BioClouZ with information of the file sent by the client; finally, (5) the client opens a connection to the destination BioNimbuZ server and sends the file.

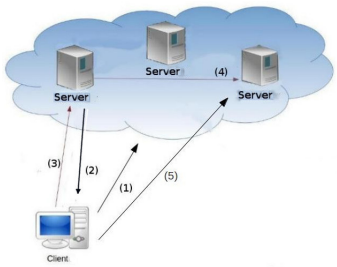


Fig. 3. An upload operation made by RPC Avro.

### 3.2 BioClouZ services

Besides uploading, other services were created to improve data management in BioNimbuZ and implemented in the BioClouZ policy:

- List: lists all the files stored in the federation. With this command, the client accesses a single list with all the files stored in BioNimbuZ, such that he/she has the illusion that these files are stored only in the server he/she is connected;
- Download: makes download to the client computer, or to the BioNimbuZ server that needs a file to perform a task. Both the download and the upload use the SFTP protocol;
- Replicate file stored in the federation: when a new file is uploaded or created in BioNimbuZ, the BioClouZ policy verifies how many copies of this file are stored in the federation. If the number of copies is less than a predefined value, the file is immediately replicated to different BioNimbus servers until reaching the number of copies set in the configuration file. Default is two, being this value easily changeable;
- Fault Tolerance: if a server leaves BioNimbuZ (e.g., due to network or energy problems), all the BioNimbuZ servers will be noticed via ZooKeeper, stating that this server is unreachable. Moreover, this service seeks, in the disconnected BioNimbuZ server (*znode*), the information of the files that became unaccessible. BioClouZ uses this information to find the BioNimbuZ servers having copies of each file stored in the disconnected server. BioClouZ send messages to these BioNimbuZ servers, warning that they should replicate these files, since one file copy has been lost due to one BioNimbuZ server crash. Thus, each server replicating this file will have to compute the storage value between itself and all the other servers, to choose the new BioNimbuZ server that will replicate each file with less one copy.

## 4 Case Studies

THREE case studies were performed to analyze the performance of the BioCloudZ storage policy in practice. They were executed with real bioinformatics data at BioNimbuZ.

For the experiments, a prototype with tree federated clouds was constructed:

- one cloud at the University of Brasilia (Brazil), composed of three machines, each with a processor Core i7-3770 3.4 GHz, 8GB RAM, 2TB hard disk, Linux operating system Ubuntu 04.13;
- one cloud at Microsoft Azure [17] (East Asia), composed of one virtual machine with 8 cores, 14GB of RAM, 30GB hard disk, Linux operating system Ubuntu 04.12 LTS;
- one cloud at Amazon EC2 [18] (United States), composed of three virtual machines, each with a porcessor Intel Xeon 2.4 Ghz, 613MB RAM, 80GB hard disk, Linux operating system Ubuntu 12.04.2.

**Case study 1.** Consisted in one upload operation of a binary file of 70 Megabytes to BioNimbuZ, with the client located in Brazil. Nine uploads were executed with the random storage policy and other nine with BioClouZ. The first BioNimbus platform executed with the random policy, just receiving the user requests and sending the files to the first BioNimbuZ server with enough space to store them.

Results showed that the BioClouZ storage policy (see Figure 4) sent the files to the cloud located in Brazil in 67% of the upload operations. In contrast, the random policy sent the files to the cloud located in East Asia in 56% of the upload operations. This shows that BioClouZ suggested to send the file to the server closest to the client, since BioClouZ considers latency the most important factor, and latency is reduced when source and destination are geographically close.

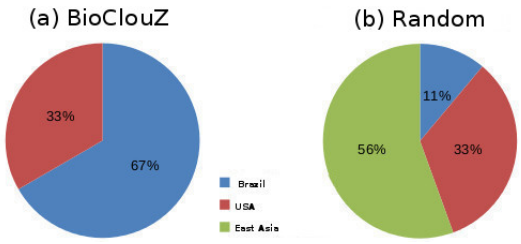


Fig. 4. File storage policy in BioNimbuZ servers. (a) BioClouZ. (b) Random.

Besides, since BioClouZ selects the closest

server, data transfer is performed faster. File download average time was 33.34 secs with BioClouZ, compared to 51.23 secs of the random policy, showing that BioClouZ reduced the transfer time in about 65%. The average transfer rate was computed from the average time of the nine file downloads, for each policy.

**Case study 2**. The objective was to verify the total execution time of jobs sent to BioNimbuZ. For this case study, a federated cloud was constructed with two clouds, one at the University of Brasilia and the other at Amazon EC2 [18], with the same configuration described before, both with BioClouZ storage policy.

We used Bowtie [4], a commonly used and typical bioinformatics tool for mapping short sequences to a reference genome. Bowtie was available as a service in BioNimbuZ. Five jobs were created, each job executing Bowtie with one input file. Table 1 shows the files with sizes ranging from 178 MB up to 252 MB each, dowloaded from the NCBI [19] database ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/Assembled_chromosomes/seq/.

TABLE 1
Sizes of the input files used to execute Bowtie.

| File name | Length |
|---|---|
| hs_alt_HuRef_chr1.fa | 252 MB |
| hs_alt_HuRef_chr2.fa | 247 MB |
| hs_alt_HuRef_chr3.fa | 198 MB |
| hs_alt_HuRef_chr4.fa | 189 MB |
| hs_alt_HuRef_chr5.fa | 178 MB |

These five jobs were sequentially executed, with the total execution time measured in milliseconds (see Figure 5). Time was computed between the initial time of the first job to the finishing time of the last job.

**Case study 3**. The objective was to investigate the influence of the number of *znodes* in the job execution time. Two executions were realized, both with two clouds (University of Brasilia and Amazon EC2, as described before, both with BioClouZ). In the first execution, only one ZooKeeper server (*znode*) was used to manage data in each BioNimbuZ server. In the second one, three ZooKeeper servers (*znodes*) were created, one at the University of Brasilia and two at Amazon EC2.

In the first execution, the total time of the five job execution was 401 seconds (see Figure 5). The inclusion of one more *znode* in Amazon EC2 led to a total time of 190 seconds, about 53% more fast. This shows that the job execution time is faster when more *znodes* are conveniently used to model each BioNimbuZ server integrating the federation.

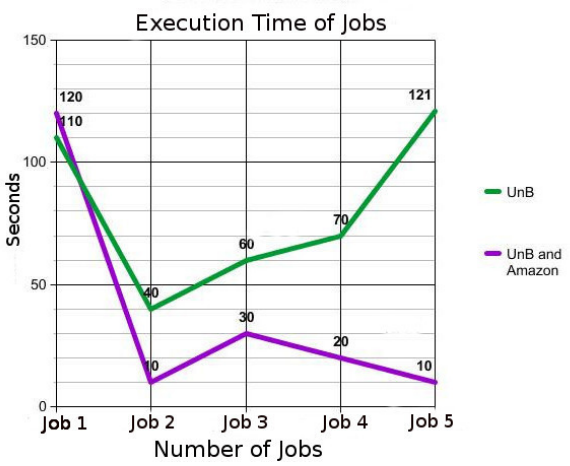


Fig. 5. Job execution times with BioClouZ.

## 5 Related Work

Recent research has been developed to design and implement generic storage infrastructure services, e.g., distributed storage services in clouds [20], [21], [22]. Many storage service providers need a strong work of integration, due to distinct interfaces and particular sets of resources provided by each provider [23], [24]. Particularly, in a federated cloud environment, execution of applications involves file storage and transferring among the clouds integrating the federation.

Although bioinformatics applications usually manipulate large volumes of data, which justifies the use of cloud environments, storage policies proposed in the literature do not consider specific characteristics of bioinformatics applications. Among these general purpose systems, many storage systems are available in the literature, e.g., Hadoop Distributed File System [25], [26], CassandraFS [27], XtreemFS [28], Lustre [29], [30] and Ceph [31].

These systems have to be carefully adapted to genomic data, in order to efficiently manage this kind of data, since gigabytes can be

easily reached in one single project. Generally, time and memory efficient bioinformatics tools execute in very large databases, requiring extensive processing and storage resources. Besides, distinct bioinformatics research institutes develop tools and generate data that may be shared. The storage policy BioClouZ, as well as BioNimbuZ, are solutions aiming to be easy to use and to provide transparency, scalability and elasticity, integrating clouds with distinct policies for data and security.

Stockinger et al. [32], in their cost model to distribute and replicate data, presented some aspects that could influence data transfer, e.g., physical location, bandwidth, size of the file to be downloaded and transmission protocol. Although this model was more concerned to cost, the authors also considered other aspects, since the objective was to transfer files in the shortest possible time, leading to a lower cost.

Zeng *et al.* [33] proposed a layer based architecture to implement a data storage service in clouds. The more important layers (metadata managing, management overlapping and interface) virtualized the details of each cloud provider, so that the user could access a more abstract level of an automatic selection management service. The BioClouZ storage policy, proposed here, is similar, since it defines a layer of virtual data management connected to each cloud integrating the federation, having also the possibility to use the local storage of these clouds. Besides, the authors used the *Ant Colony Optimization* (ACO) algorithm.

Yuan [34] proposed an algorithm to select a cloud as the data storage provider, using a grouping strategy based on the *k-means* model.

Ruiz-Alvarez and Humphrey [35] devised an automatic mechanism to select the storage management. The main contributions of this work were: (i) an XML schema to describe the cloud providers and respective storage resources; and (ii) an algorithm to select an storage service based in criteria supplied by the users, e.g., cost, data transfer, durability and data versioning. A key characteristic of their mechanism is the possibility to access statistics of the storage providers in real time, so that the algorithms can use current information to decide the best location to store data.

Bermbach et al. [36] reported the problem of the dependency of one single storage service, which could limit availability and scalability, besides causing delays in executions carried out by the client's service. To minimize this problem, they proposed MetaStorage, a federated storage system using hash tables integrating different suppliers through data replication.

Zhang and Zhang [7] proposed a model also based on layers (access, applications and storage infrastructure), through a P2P communication. The infrastructure layer is responsible for carrying out the distribution of files between two different storage services, channels and devices, as well as for performing compression and redundancy. The storage layer controls the processes to be executed. Updating and retrieving, among others, ensure information availability.

Research in cloud storage and data compression usually consider large amounts of data. Nicolae [37] presented BlobSeer, a model to manage distributed data, specially implemented to read, write and collate large amounts of information on a large scale, also performing data compression. Bandwidth is used to transfer data. This approach allows to decrease the amount of disk space. However, it should consider also feasibility due to both time and cost required to perform such operations.

Recently, Hiden et al. [38] presented a model that allows to execute workflows and share applications developed in distinct institutions (similarly to BioNimbus [8]), where the storage management is hidden by some services. However, their structure can not be used in hybrid clouds, like BioNimbuZ.

A more detailed investigation about data storage in cloud federation is still necessary. In this context, two tendencies in storage policies have to be studied, integration of data processing requirements as well as other mechanisms, e.g., security and criptography [39], [40].

## 6 Conclusion

In this paper we presented BioClouZ, a storage policy to the BioNimbuZ hybrid federated cloud platform, which considers factors such as computational capacity of the resources, type of clouds (charged or free data)

and services provided by each federated cloud. Three case studies were performed, and results showed that BioClouZ led to a more efficient storage policy, in the sense that the total time to execute a bioinformatics application is lower when comparing to the same application executing with a random storage policy, previously used in BioNimbuZ. Moreover, replication and fault tolerance operations improved data availability as well as file retrieval both to the clients and the federated clouds. To implement the novel storage policy, updates in the first BioNimbus platform were performed, leading to a second version called BioNimbuZ. Using Zookeper and Avro, the communication among the federated clouds was changed, and a decentralized coordination was developed.

Other bioinformatics applications can be easily adapted to BioNimbuZ, since the platform was designed to be independent from particular tools and databases. Each cloud entering in the federation may offer its services, mantaining its security and storage policies, and also can benefit from the services offered by the other clouds. Therefore, commonly used bioinformatics tools, e.g., Blast and HMMER/Pfam, as well as workflows, may be easily ported to BioNimbuz, if clouds entering in this federated platform offer them as services.

Despite the good results obtained by BioClouZ, it could be improved by keeping local and current parameters to foresee the better clouds to store and replicate data. Besides, the computation of storage values of the clouds, based on four parameters (free space size, uptime, latency and storage cost), could include actual compute capacity of the virtual machines where data will be processed. In this direction, we intend to measure to what extent moving data, e.g., to a 64 Gb machine in Asia affects the latency of moving data to a 2 Gb machine at the University of Brasilia, since the resource capacity of the machines executing one application could affect the total execution time. Experiments with larger files and more machines on each cloud are planned. Moreover, the implementation of P-FTP (Parallelized File Transfer Protocol) [41] to a federated cloud environment could accelerate data transfer between clients and servers. Finally, regarding to security aspects, implementation of an ACL (Access Control List) associated to the files stored in the federation will assure that only authorized users could use and modify these files. Clouds where data can be stored and other data security information could be negotiated between users and BioNimbuZ.

## ACKNOWLEDGMENTS

A. Araujo, M. Holanda and M.E. Walter would like to thank STIC-AmSud project (CAPES 41/2013) and M. E. Walter would like to thank CNPq (Project 308509/2012-9).

## REFERENCES

[1] S. V. Angiuoli, M. Matalka, A. Gussman, K. Galens, M. Vangala, D. R. Riley, C. Arze, and J. R. White, "CloVR: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing," *BMC Bioinformatics*, vol. 12, no. 356, pp. 1–15, 2011.

[2] S. V. Angiuoli, J. R. White, M. Matalka, O. White, and W. F. Fricke, "Resources and costs for microbial sequence analysis evaluated using virtual machines and cloud computing," *PLoS ONE*, vol. 6, no. 10, p. e26624, 2011.

[3] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. Salzberg, "Searching for SNPs with cloud computing," *Genome Biology*, vol. 10, no. 11, p. R134, 2010.

[4] 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, pp. R25+, 2009.

[5] B. Pratt, J. J. Howbert, N. Tasman, and E. J. Nilsson, "MR-Tandem: parallel X!Tandem using Hadoop MapReduce on Amazon Web Services," *Bioinf.*, vol. 8, pp. 1–12, 2011.

[6] M. C. Schatz, "CloudBurst: Highly sensitive read mapping with MapReduce," *Bioinformatics*, vol. 25, pp. 1363–1369, 2009.

[7] L. Zhang, S. Gu, B. Wan, Y. Liu, and F. Azuaje, "Gene set analysis in the cloud," *Bioinform.*, vol. 13, pp. 1–10, 2011.

[8] H. Saldanha, E. Ribeiro, C. Borges, A. Araujo, R. Gallon, M. Holanda, M. E. Walter, R. Togawa, and J. C. Setubal, "Towards a hybrid federated cloud platform to efficiently execute bioinformatics workflows," in *BioInform.*, H. Pérez-Sánchez, Ed. In Tech, 2012, ch. 5, pp. 107–132.

[9] Zookeeper, "The Apache Software Foundation," 2013, http://zookeeper.apache.org/.

[10] Avro, "Apache Avro$^{TM}$ 1.7.4 Documentation," 2013, http://avro.apache.org/docs/current/.

[11] L. Wu and R. Buyya, "Service Level Agreement (SLA) in Utility Computing Systems," Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Tech. Rep. CLOUDS-TR-2010-5, 2010.

[12] D. Crockford, "The application/JSON Media Type for JavaScript Object Notation (JSON)," 2006, rFC 4627 (Informational). http://tools.ietf.org/search/rfc4627. JSON: http://json.org/.

[13] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[14] P. Maymounkov and D. Maziéres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK: Springer-Verlag, 2002, pp. 53–65.

[15] H. Verespej and J. Pasquale, "A characterization of node uptime distributions in the PlanetLab test bed," in *30th IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2011, pp. 203–208.

[16] R. Xun-Yi and M. Xiao-Dong, "A* algorithm based optimization for cloud storage," *JDCTA*, vol. 4, no. 8, pp. 203–208, 2010.

[17] Azure, "A solid cloud platform for creative ideias," 2013, http://www.windowsazure.com/pt-br/.

[18] Amazon, "Amazon Elastic Compute Cloud," 2013, http://aws.amazon.com/pt/ec2/.

[19] NCBI, "U.S. National Library of Medicine, National Center for Biotechnology Information," http://www.ncbi.nlm.nih.gov/, 2013.

[20] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "BigTable: A distributed storage system for structured data," in *Proc. 7th Conference on Usenix Symposium on Opertating Systems Design and Implementation, volume 7*, 2006, pp. 205–218.

[21] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[22] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *SOSP*, vol. 7, 2007, pp. 205–220.

[23] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *Proc. 10th ACM SIGCOMM Conf. on Internet Measurement*, 2010, pp. 1–14.

[24] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, "Early observations on the performance of Windows Azure," in *ACM Proc. 19th International Symposium on High Performance Distributed Computing*, 2010, pp. 367–376.

[25] D. Borthakur, "Hadoop distributed file system: architecture and design," *Hadoop Project Website*, pp. 1–14, 2007, http://hadoop.apache.org/common/docs/r0.18.0/hdfs_design.pdf.

[26] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proc. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[27] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[28] F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario, "The XtreemFS architecture—a case for object-based file systems in grids," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 17, pp. 2049–2060, Dec. 2008.

[29] Lustre, "Lustre file system," http://lustre.opensfs.org/, 2013.

[30] J. Piernas, J. Nieplocha, and E. J. Felix, "Evaluation of active storage strategies for the lustre parallel file system," in *Proc. 2007 ACM/IEEE Conference on Supercomputing*, ser. SC '07, New York, NY, USA, 2007, pp. 28:1–28:10.

[31] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symposium on Operating Systems Design and Implementation (OSDI)*, 2006, pp. 307–320.

[32] H. Stockinger, K. Stockinger, E. Schikuta, and I. Willers, "Towards a cost model for distributed and replicated data stores," in *Proc. Nineth Euromicro Workshop on Parallel and Distributed Processing*. Wien Univ., Austria: IEEE Computer Society, 2001, pp. 461–467.

[33] W. Zeng, Y. Zhao, K. Ou, and W. Song, "Research on cloud storage architecture and key technologies," in *ACM Proc. 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, 2009, pp. 1044–1048.

[34] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010.

[35] A. Ruiz-Alvarez and M. Humphrey, "An automated approach to cloud storage service selection," in *Proc. 2nd international workshop on Scientific cloud computing*. ACM, 2011, pp. 39–48.

[36] D. Bermbach, M. Klems, S. Tai, and M. Menzel, "Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs," in *Proc. 2011 IEEE 4th International Conference on Cloud Computing*, ser. CLOUD '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 452–459.

[37] B. Nicolae, "High throughput data-compression for cloud storage," in *Proc. Third international conference on Data management in grid and peer-to-peer systems*, ser. Globe'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 1–12.

[38] H. Hiden, S. Woodman, and P. Watson, "Developing cloud applications using the e-science central platform," *Philosophical Transactions of the Royal Society - A Mathematical Physical Engineering Sciences*, vol. 371, no. 1, pp. 17–32, 2013.

[39] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *Security & Privacy, IEEE*, vol. 8, no. 6, pp. 24–31, 2010.

[40] T. Mather, S. Kumaraswamy, and S. Latif, *Cloud security and privacy: an enterprise perspective on risks and compliance*. O'Reilly, 2009.

[41] S. Sohail, S. Jha, and H. ElGindy, "Parallelized file transfer protocol (P-FTP)," in *Proc. 28th Annual IEEE International Conference on Local Computer Networks (LCN'03)*, 2003, pp. 624–631.