Chapter 5

Towards a Hybrid Federated Cloud Platform to Efficiently Execute Bioinformatics Workflows

Hugo Saldanha, Edward Ribeiro, Carlos Borges, Aletéia Araújo, Ricardo Gallon, Maristela Holanda, Maria Emília Walter, Roberto Togawa and João Carlos Setubal

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/50289

1. Introduction

Current generation of high-throughput DNA sequencing machines [1, 35, 66] can generate large amounts of DNA sequence data. For example, the machine HiSeq 2000 from the company Illumina, a current workhorse of genome centers, is capable of generating 600 Giga base-pairs of sequence in one single run [35]. The Human Microbiome project (https://commonfund.nih.gov/hmp) and the 1000 Genomes project (http://www.1000genomes.org) are two examples of projects that are generating terabyte-scale amounts of DNA sequence.

Such vast amounts of data can only be handled by powerful computational infrastructures (also known as cyberinfrastructures), sophisticated algorithms, efficient programs, and well-designed boinformatics workflows. As a response to this challenge, a large ecosystem composed by different technologies and service providers has emerged in recent years with the paradigm of cloud computing [2, 58, 63, 71]. In this paradigm users have transparent access to a wide variety of distributed infrastructures and systems. In this environment, computing and data storage necessities are accomplished in different and unanticipated ways to give the user the illusion that the amount of resources is unrestricted.

In this scenario, cloud computing is an interesting option to control and distribute processing of large volumes of data produced in genome sequencing projects and stored in public databases that are widespread in distinct places. However, considering the constant growing of computational and storage power needed by different bioinformatics applications that are continously beeing developed in different distributed environments, working with one single cloud service provider can be restrictive for bioinformatics applications. Working with more than one cloud can make a workflow more robust in the face of failures and unanticipated needs. Cloud federation [11, 14, 15] is one such solution. Cloud federation offers other advantages over single-cloud solutions. Bioinformatics centers can profit from participation in a cloud federation, by having access to other center programs, data, execution and

storage capabilities, in a collaborative environment. The federation can abstract cloud-specific mechanisms, thus potetially making the use of such a resource more user-friendly and easier to install and customize. This is particularly valuable for small and medium centers that can enlarge their hardware resources and software tools using machines and programs of other centers integrating a federated system.

In this work, we propose a hybrid federated cloud computing platform that aims at integrating and controlling different bioinformatics tools in a distributed, transparent, flexible and fault tolerant manner, also providing highly distributed processing and large storage capability. The objective is to make possible the use of tools and services provided by multiple institutions, public or private, that can be easily aggregated to the cloud. We also discuss a use case of this platform, a bioinformatics workflow for identifying differentially expressed genes in cancer tissues.

2. Federated cloud computing

There are many distinct definitions of cloud computing. According to [29], cloud computing could be defined as "*a computational paradigm highly distributed, directed by a scale economy, in which the computational power, storing, abstract platforms and services, virtualized, managed and dinamically scalable are provided on demand by external users through the Internet*".

[71], using all the characteristics collected from the literature, proposed a definition of clouds as "*a big pool of virtualized resources, easily usable. The resources can be reconfigured dinamically according to a variable load, allowing optimized using. This pool is typically explored by a pay-per-use model in which guarantees are offered by the infrastructure provider, following a service contract*". These authors attempted to define cloud computing using only common characteristics in cloud providers, but they did not find features that were mentioned by all providers. The most common were scalability, pay-per-use model and virtualization.

From these definitions we can state that the goal of cloud computing is to offer to users the idea that they have unrestricted resources, but they have to pay only for those effectively used (model pay-per-use). Another significant advantage of clouds is the management of the computational infrastructure, relieving users from concerns such as power failures and backups. The property of allocating computational resources depending on user needs is called elasticity.

Cloud services can be deployed by providers in different ways [48]:

- Private cloud: operated for the use of a single organization. It can be managed by the organization itself or by external ones.
- Community Cloud: shared by several organizations and used as a tool for a specific group of users with common interests.
- Public Cloud: available to the general public or a large corporate group that is part of the organization that sells this service.
- Hybrid cloud: composed of two or more clouds (private, community or public) that remain separate entities, but that are bound together by standardized or proprietary technologies that enable portability of data and applications.

In clouds, one of the key technologies adopted to execute bioinformatics programs is the Apache Hadoop framework [6], in which the MapReduce [25] model and its distributed file system (HDFS) [13] are used as infrastructure to distribute large scale processing and data storage. In the MapReduce model parallelization does not require communication among simultaneously processed tasks, since they are independent from one another.

Bittman [11] claimed that the evolution of cloud computing market could be divided in three phases. In phase 1 (Monolithic), cloud computing services were based on proprietary architectures, or cloud services were delivered by megaproviders. In phase 2 (Vertical Supply Chain), some cloud providers leveraged services from other providers, i.e. independent software vendors (ISVs) developed applications as a service using an existing cloud infrastructure. Clouds were still proprietary, but ecosystems construction started. In phase 3 (Horizontal Federation), smaller providers would horizontally federate to gain economy of scale and efficient use of their assets. Projects would leverage horizontal federation to enlarge their capacibilities, more choices at each cloud computing layer would be provided, and discussion about standards would begin.

In general, cloud computing intends to increase efficiency in service delivery, dealing with services including infrastructure, platforms and software, and treating with distinct users like a single user, other clouds, academic institutions and large companies. Besides public clouds maintained by large organizations, hundreds of smaller heterogeneous and independent clouds, private or hybrid, are being developed. In this scenario, cloud federation becomes an interesting way to optimize the use of the resources offered by various organizations. In particular, in this chapter, we are interested in horizontal cloud federation, also called federated cloud computing, inter-cloud [14] or cross-cloud [15].

Federated cloud computing can be defined as a set of cloud computing providers, public and private, connected through the Internet [14, 15]. Among its objectives we distinguish the seemingly availability of unrestricted resources, independence of a single infrastructure provider, and optimization when using a set of distinct resource providers.

Thus, federation allows each cloud computing provider to increase its processing and storage capabilities by requesting more resources to other clouds in the federation when needed. This means that a local cloud provider is able to satisfy user requests beyond its capabilities, since idle resources from other providers can be used. Furthermore, if a provider fails, resources can be requested to another one, providing more fault tolerance.

Although the advantages of federated cloud computing are obvious, its implementation is not trivial, since the participating clouds present heterogeneous and frequently changing resources. Therefore, traditional models of federation are not useful [15]. Typically, federated models are based on *a priori* agreements among their members, noting that these agreements can be inappropriate according to the particular characteristics of a cloud provider. Thus, to make possible the creation of a federated cloud environment, it is necessary to achieve the following requirements [14, 15]:

- **Automatism**: a cloud member of the federation, using discovery mechanisms, should be able to identify the other clouds in the federation together with their resources, responding to changes in a transparent and automatic way;

- **Application behavior prediction**: the system implementing the federation has to be able to predict demands and behaviors of the offered applications, so that its load balancing mechanism can have its efficiency improved;
- **Mapping services to resources**: the services offered by the federation must be mapped to available resources in a flexible manner so that it can achieve the highest levels of efficiency and cost/benefit. In other words, the schedule must choose the best hardware-software combination to ensure the quality of service at lowest cost, taking into account the uncertainty of the availability of resources;
- **Interoperable security model**: federation must allow the integration of different security technologies so that a cloud member does not need to change its security policies when entering the federation;
- **Scalability in monitoring components**: considering the possible large number of participants, the federation must be able to handle multiple task queues and the largest number of requests, so that management can guarantee that the various cloud providers of the federation will mantain scalability and performance.

It is noteworthy that issues to choose an appropriate cloud provider and lack of common cloud standards hinder the interoperability across these federated cloud providers. Thus, nowadays the user is faced with the challenging problem of selecting the appropriate cloud that fits his or her needs. To address this problem, the BioNimbus platform offers to users a federated platform that can execute bioinformatics applications in a transparent and flexible manner. This is possible because BioNimbus offers standardized interfaces and intermediate services to manage the integration of different cloud providers. Moreover, as will be seen next, BioNimbus was designed to incorporate the requirements defined by [15].

3. BioNimbus: a federated cloud platform

As mentioned before, cloud computing is a promising paradigm for bioinformatics due to its ability to provide a flexible computing infrastructure on demand, its seemingly unrestricted resources, and the possibility to distribute execution in a large number of machines leading to a significant reduction in processing time due to the high degree of achieved parallelism. Some bioinformatics tools have been implemented in cloud environments belonging to several infrastructures of physically separated institutions, which makes it difficult for them to be integrated.

BioNimbus [12, 62] is a federated cloud platform designed to integrate and control different bioinformatics tools in a distributed, flexible and fault tolerant manner, providing rapid processing and large storage capabilities, transparently to users. BioNimbus joins physically separate platforms, each modelled as a cloud, which means that independent, heterogenous, private/public clouds providing bionformatics applications can be used as if they were a single system. In BioNimbus, resources of each cloud can be maximally explored, but if more are required, other clouds can be requested to participate, in a transparent manner. BioNimbusis thus able to satisfy further service allocation requests sent by its users. The objective is to offer an environment with apparently unrestricted computational resources given that computing and storage space demands are always provided on demand to the users.

3.1. BioNimbus architecture

All the components of BioNimbus architecture together with their funcionalities are defined such that it allows simplicity, speed and eficiency when a new cloud provider enters in the federation. Another key characteristic is the communication among the BioNimbus components that is realized through a Peer-to-Peer (P2P) [67] network, guaranteeing the following properties:

- Fault tolerance, since there is not a single fail point. Thus, even if some nodes fail, the others can work;
- Efficiency, since there is not a single bottleneck. Then, messages are end-to-end and not routed by a single node;
- Flexibility, since clouds can operate independently or in a coordinated manner;
- Scalability, since the use of a P2P network allows integration of thousands of interconnected machines.

BioNimbus (Figure 1) architecture enables the integration of different cloud computing platforms, meaning that independent, heterogeneous, private or public providers may offer their bioinformatics services in an integrated manner, while maintaining their particular characteristics and internal policies. BioNimbus is composed of three layers: application layer, core layer and cloud provider layer.

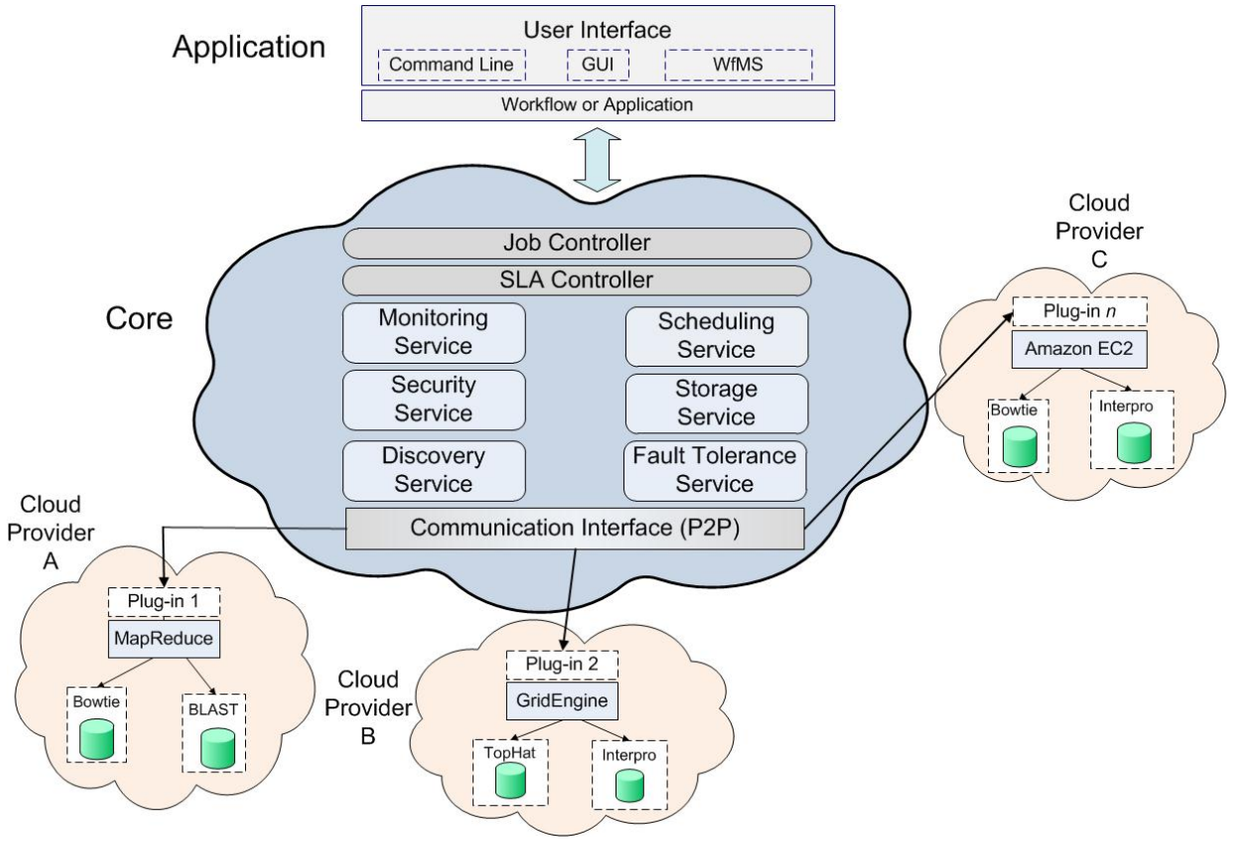

Figure 1. The architecture of BioNimbus hybrid federated cloud.

3.1.1. Application layer

This layer provides the service of interaction with users, which can be implemented by web pages, command lines, graphical interfaces (GUI) or workflow management systems (WfMS). Users can execute workflows or a single application, choosing among available services. Job controller service has the function of collecting user requests and sending the input data to the core layer. Moreover, this layer is responsible for showing to each user the current status of his running applications. Users can list the files stored in the federation, upload or download files, create and execute workflows in the BioNimbus cloud environment.

3.1.2. Cloud provider layer

This layer encompasses the cloud providers belonging to BioNimbus. The previous described core layer ensures a unified view of the cloud, which allows users to see all the resources available on each cloud as if they were one unified system.

A plug-in service is used to integrate a cloud provider (public or private) in the federation. Each plug-in service is an interface that aims at communicating the BioNimbus core with each cloud provider. Cloud providers can communicate among themselves also using the core layer. Each plug-in needs to map the requests sent by the core components to the corresponding actions that have to be realized in each cloud provider. This implies that each cloud requires a special plug-in service. Furthermore, to integrate distinct providers (public or private), each plug-in needs to treat three different kinds of requests: information about the provider infrastructure, task management and file transfer.

3.2. Core layer

This layer is responsible for managing the federated environment. Among their functions are: identification of new providers with their corresponding hardware and software resources; task scheduling and controlling; definition, establishment and monitoring of SLA (Service Level Agreement); storage and managing of input and output files; maintainance of the online environment; and the election of new coordinators for each requested service. To each function implemented in this layer, a controller service was included in the architecture, as described next.

3.2.1. Discovery service

This service identifies the cloud providers integrating the federation, and consolidates information about storage and processing capabilities, network latency, availability of resources, available bioinformatics tools, details of parameters and input and output files. To realize this, the discovery service waits for information published by providers about their infrastructure and available tools. To consolidate these data, the discovery service maintains a data structure that is updated whenever new data is received. Furthermore, the discovery service has a policy of controlling each provider, removing from the federation those providers not regularly sending updated information, which guarantees the correct and update task execution on the federated cloud. Regarding to the entrance of a cloud provider in BioNimbus, *a priori*, at any time a peer participating in the P2P network can start the process of publishing its resources (storage and processing capabilities) and available bioinformatics applications. However, for security and controlling purposes, permission to

join the federation as a provider must be verified with the support of the security service whenever any information about a new provider arrives to the discovery service.

As can be seen from the above description, an efficient resource discovery mechanism plays a central role in our federated cloud, since the information gathered by this service is essential to other services to properly perform their functions. According to [49], in large-scale distributed services, a resource discovery infrastructure has to meet the following key requirements: it must be scalable so it can handle thousands of machines without being unavailable or losing performance; it must be able to handle both static and dynamic resources; and it must be flexible enough so its queries could be extended in order to handle different types of resources. Possible implementations of a resource discovery service could be developed using central or hierarchical approaches, but these are known to have serious limitations of scalability, fault-tolerance and network congestion [56].

In BioNimbus, we plan to use a publish/subscribe mechanism, in which providers publish information about their resources to a decentralised resource discovery system. This system will use a Distributed Hash Table (DHT) data structure [7] in order to achieve low management costs and network overhead, efficient resource searching and fault-tolerance. For resource information handling, we plan to use serializable and extendable formats such as the JSON format [23]. In this way it will be possible for the federated cloud to deal with different types of information, thus causing the least impact possible.

3.2.2. Job controller

The job controller links the core and the application layers of BioNimbus. It first calls the security service to verify if a user has permission (authentication) to execute jobs in BioNimbus and what are the credentials of this user. Moreover, the job controller's main function is to manage distinct and simultaneously running workflows, noting that the workflows may belong to the same or to different users. Thus, for each accepted workflow, the job controller generates an associated ID and controls each workflow execution using this ID.

3.2.3. SLA controller

According to [74], SLA is a formal contract between service providers and consumers to guarantee that consumers' service quality expectations can be achieved. In BioNimbus, the SLA controller is responsible for implementing the SLA lifecycle, which has six steps: discovers service providers, defines SLA, establishes agreement, monitors SLA violation, terminates SLA and enforces penalties for violation. A SLA template represents, among others, the QoS parameters that a user has negotiated with BioNimbus. The user can populate, through an user interface, a suitable template with required values or even define a new SLA from scratch in order to describe functional (e.g. CPU cores, memory size, CPU speed, OS type and storage size) and non-functional (e.g. response time, budget, data transfer time, availability and completion time) service requirements. In bioinformatics, functional requirements are number of cores, amounts of memory and storage, CPU speed, bioinformatics programs and databases and respective versions. Non-functional requirements are latency (transfer rate) and uptime (reliability, in the sense that it measures how frequently a cloud provider is running tasks or if it is not entering and leaving the federation).

The SLA controller has the responsibility to investigate whether the SLA template submitted by the user can be supported by the federated cloud platform. For this, the SLA controller

retrieves the SLA level published through the provider plug-in (e.g. gold, silver or bronze SLA level).

So, if the service agreement required by the SLA template can not be satisfied, a negotiation phase starts. The SLA negotiation phase is done as follows: the user submits a service request with the new SLA template to the job controller. Next, after parsing the SLA definition, the SLA controller asks the monitoring service if it could execute the service with the specified requirements. In order to respond to this request, the monitoring service requests the scheduling service to find the best suitable provider by matching the gathered resource properties to the service requirements by applying predefined scheduling algorithms. If none of the providers can be matched, the monitoring service enables the discovery service, which must seek new cloud providers to be integrated into the federated environment, aiming at satisfying the SLA template requested by the user. However, if this is not possible, the mentioned steps must be repeated for renegotiation, with a new SLA template, until reaching an agreement.

After establishing an agreement, the SLA controller generates an ID for the agreement, and sends the ID to the job controller, which records this ID agreement. Then, the job controller forwards both the request and the agreement ID to the monitoring service, which sends the tasks for the scheduling service. The monitoring service is responsible for checking if a violation of the agreement occurred and in this case it immediately has to inform the SLA controller, which terminates the SLA and enforce penalties for violation.

3.2.4. Monitoring service

This service verifies if a requested service is available in a cloud provider, searching for another cloud in the federation if it is not; receives the tasks to be executed from the job controller, and sends them to the scheduling service that will distribute them, guaranteeing that all the tasks of a process are correctly executed; and informs the job controller when a task successfully finishes its execution. To ensure the monitoring of all the requested tasks, this service periodically sends messages to the clouds that are executing tasks, and informs the user the current status of each submitted task.

To perform the activities described above, the monitoring service must be able to gather information about resource allocation and task execution, which depends on the application being executed [28]. Therefore, we have to establish some criteria about the frequency that data are obtained and their corresponding format, so that the decision-making process performed by this service can be made with reliability with respect to data timeliness and flexibility towards distinct applications. In BioNimbus, the monitoring service was planned to send messages at regular intervals to all the federation members or whenever needed. The latter case happens when a decision has to be taken for a specific federation member or when data update is necessary. All information exchange is done with timestamps so only the updated data are sent in order to save network bandwidth. We also plan to use an extensible and flexible format, such as JSON [23], like in the discovery service.

In federated clouds the monitoring service must have other characteristics, such as: scalability, to handle a large number of resources and tasks to be monitored; elasticity, to handle addition and removal of resources in a transparent manner; and federation, to handle entering and leaving providers [20]. In order to meet these requirements, we propose to use a decentralized information indexing infrastructure, which would be the same DHT available to the discovery

service. Furthermore, as previously described, the monitoring service has to verify agreement violations.

3.2.5. Storage service

This service decides how to distribute and replicate data among the cloud providers integrating the federation [8, 39, 73], particularly model the storage strategy of the files consumed and produced by the jobs executed in BioNimbus. To realize this, the storage service can communicate with the discovery service to access information about the federation, since the discovery service knows the actual storage conditions of each provider integrating the federation. Thus a storage policy is defined, so that this choice can be made based on receiving information about the file (FileInfo) and returns at least one cloud provider (PluginInfo) to store this file.

Some characteristics of biological data for bioinformatics applications are: large volume; it is not necessary to guarantee the ACID transaction properties since there are no users simultaneously updating data during execution; according to a particular bioinformatics application, fragmentation and replication can use different models; and data provenance is essential.

Considering these characteristics, the BioNimbus storage service proposal is based on the HBase NoSQL (Not only SQL) database. Among distinct noSQL databases, like HBase [38], Dynamo [26], Bigtable [17], Cassandra [41], PNUTS [22], monogDB [19] and cloudDB [5], we adopted HBase since its basic data storage is Apache's Hadoop Distributed File System (HDFS) [13] and it is a column-oriented database [24, 50, 51], which allows joining data (biological data file) and sets of information (e.g. data provenance).

Replication will be done copying data to at most three clouds in order to ensure recovery in case of failure. Total or partial fragmentation depends on the biological data and the application. On the top of HBase, we propose to create an Analyzer module, which will decide where the replication has to be done. The objective of the Analyzer module is to reduce data transfers among the cloud providers, based on three criteria: disk space, geographic position and data transfer speed. The most important analysis is the available space, which should be sufficient to store input and output. The geographic position criterion has the objective of reducing data transfer on the network, being the closer clouds used first if possible. Finally, the Analyzer module uses data transfer speed, which must be computed using the time to transfer packages.

3.2.6. Security service

This service guarantees integrity among the distinct tasks executed in the federated clouds. A federated cloud needs to include the security policies of each cloud provider while avoiding strong inter-dependency among the clouds. A security context can be partitioned into three main topics: authentication, authorization and confidentiality. We address those requirements using standard algorithms and protocols as described next.

- **Authentication**: The descentralized federated cloud infrastructure should not make a centralized authentication, which is a not a good choice because it limits the scalability and

creates a strong interdependency among the clouds. We intend to use a Single Sign-On (SSO) protocol [52] so that no central authority is in charge of its users' authentication, which prevents a single point of failure and allows scalability according to the number of users. We chose the OpenID standard [57] as our SSO mechanism, since it has been used by corporate and academic sites around the world. OpenID allows each "site" (e.g. a cloud) to provide an authentication facility to its users so that they do not need to authenticate with each other cloud integrating the federation. Instead, each cloud provider acts as an identity provider for user credentials, so that each user should authenticate with its affiliated provider. Once this user is authenticated, each time his/her credentials are required, OAuth [10] allows a user's site to forward authorization without exposing the user account or login information.

- **Authorization**: The authorization of a federated cloud resource is provided by the Access Control Lists (ACLs) [69] provided by each cloud provider. An ACL determines who can access a given resource, e.g. disk storage, CPU cycles and bioinformatics services. Therefore, each cloud is able to determine access patterns so that it can control its resource's uses.
- **Confidentiality**: Communication between each two cloud providers is established using TLS/SSL [68] connection. The use of secure connections between two clouds in the federation is not enforced by our model, but it can be provided as well. As far as we know, few cloud systems provide secure intra-cloud communication. Each cloud should provide a certificate that will be used by hosts in two clouds to establish a secure connection. As we improve BioNimbus, we plan to include audit trails so that each required resource can be available when needed.

3.2.7. Fault tolerance service and high availability

This service guarantees that all the core services are always available. In a cloud environment, machine failures occur, and it is well known among the cloud community that those failures are the norm rather than the exception. Thus, any federated cloud should be designed for fault recovering and system availability. Therefore, a fault tolerance service is an essential part of our federated cloud, and has the objective of providing high availability and resiliency against periodic or transient failures.

There are extensive studies in the literature on failure detection systems [16, 31, 45, 70]. On the other hand, few systems are designed to scale with a large number of nodes as those found on clouds. Thus, an important requirement of our fault detection service is to be scalable with a large number of machines. We adopted a modified gossip based failure detector proposed by Renesse et al [70], which works as described. Each host runs the gossip failure detector service, which maintains a list of known hosts in the cloud. Every T_{seconds}, a host increases a heartbeat, and at random chooses a set of nodes for sending a list of known nodes. When received, each list is merged with the host current list, assuming the largest heartbeat for each node in the list. If a node does not update its heartbeat for a T_{elapsed} time, then it will be marked as failed. Note that a node may be marked as failed due to slow network links or even in presence of a fractioned network. But our failure detection service is conservative so that it only purges a host from the list after a $T >= 2 * T_{\text{elapsed}}$.

Besides using this gossip based failure detector, we use a coordination service based on atomic broadcast protocol [59]. The open source system Apache Zookeeper [34] runs on each cloud and allows our system to detect node failures and realize an election of leaders among the cloud machines, in order to guarantee the services availability, including discovery and fault tolerance services. Zookeeper is used to elect some of the nodes that are known as gossip servers. Those servers are dinamically chosen so that they can exchange the list of nodes among the cloud providers. This helps to reduce the bandwidth between two clouds to a few servers.

3.2.8. Scheduling service

This service dynamically distributes tasks among the cloud providers belonging to the federation, maintaining a register for the allocated tasks, controlling load of each cloud provider, and redistributing the tasks when resources are overloaded. The scheduling service is responsible for receiving the tasks created from the user requests, and maintaining a record about the status of each executed task. Before being executed in a cloud provider, a task is sent to the scheduling service, which uses one or more scheduling policies to choose the cloud provider that will execute this task, according to the negotiated SLA. Each policy receives a list of tasks to be scheduled and an agreement ID, and returns a mapping of the tasks and the cloud providers where these tasks will be executed. To do this, the scheduling policy communicates with the discovery service. The scheduling policy should consider the SLA QoS parameters and the margin values accepted by the cloud providers (e.g. gold, silver and bronze SLA level). These parameters are important for a matching of a cloud provider that is done by the scheduling policy, and therefore the user needs to give reasonable values for them. Some typical SLA parameters used in context of a cloud provider are CPU cores, CPU speed, memory size, in/ou bandwidth, OS type, storage size, response time, budget, data transfer time, completion time and availability. In BioNimbus, the scheduling service can be easily modified to use different scheduling policies.

We implemented a new *DynamicAHP* algorithm in BioNimbus [12]. The key idea of DynamicAHP is to map available resources of the cloud providers to the requested tasks, then associating a cloud to execute each task. This algorithm is based on a decision making strategy proposed by [61]. DynamicAHP worked well on a first BioNimbus prototype, since it was capable to dinamically scale using only the knowledge about the length of each task input file, while performing load balancing among the cloud providers. Since BioNimbus stores information about the cloud providers such as network latency and wait time in the execution queues, DynamicAHP reduced costs and execution time of the tasks. The promising results obtained from developing DynamicAHP in BioNimbus showed that good scheduling algorithms can really lower the time to execute bioinformatics applications in federated clouds.

It is interesting to investigate new scheduling metrics, mainly related to costs. For example, a public cloud can be associated to a lower priority due to its associated costs, when compared to other public clouds integrating the federation. Another idea is to assign weights to the metrics that could set a priority order among them. To develop and analyze a model capable of storing information about the executed tasks, such that the scheduling service could combine this information to estimate the execution time of a particular task is another challenging project.

3.3. Performing tasks in BioNimbus

Figure 2 shows how BioNimbus works. Initially (step 1), the user interacts with BioNimbus through an interface, which could be a command line or a web interface, for example. The user informs details of the application (or workflow) to be executed, and these information are sent to the job controller in form of jobs to be executed. Then, the job controller verifies the availability of the informed applications and input files, sending a response message to the user accordingly. Afterwards, these jobs' features are analyzed by the security service (step 2), which verifies the user permission to access the resources of the federation, and sends a response to the job controller (step 3).

If the requested jobs can be executed, a message is sent to the SLA Controller (step 4) that investigates whether the SLA template submitted by the user can be identified by BioNimbus. If the user request can be executed, the SLA controller sends a message to the monitoring service (step 5), which stores the jobs in a pending task list. This service is responsible for informing to the scheduling service that there are pending jobs waiting to be scheduled.

Next (step 6), the scheduling service starts when the monitoring service informs that there are pending jobs. The scheduling policy adopted in BioNimbus can be easily changed, according to the characteristics of a particular application. The scheduling service gets information about the resources using the discovery service (steps 9 and 10), which periodically updates the status of the federation infrastructure, and stores these information in a management data structure. This information is used to generate the list of ordered resources, and to assign the more demanding jobs to the best resources, according to the scheduling policy.

With the resource and job ordered lists, the scheduling service communicates with the storage service to ensure that all the input files are available to the providers chosen to execute the jobs (steps 7 and 8).

Next, the scheduler distributes instances of jobs (tasks) to be executed by the plug-ins and their corresponding clouds (steps 11 and 12).

The scheduling service decision is then passed to the monitoring service (step 13) so that it can monitor each job status until it is finished. When the jobs are all completed, the monitoring service informs the SLA Controller (step 14), which sends a message to the job controller (step 15). Finally, the job controller communicates with the user interface (step 16) informing that the jobs were completed, which closes one execution cycle in BioNimbus.

The BioNimbus architecture follows [3], who claims that high-throughput sequencing technologies have decentralized sequence acquisition, which increases demands for new and efficient bioinformatics tools that have to be easy to use, portable across multiple platforms, and scalable for high-throughput applications.

4. A case study

A federation with two cloud providers, one nonpublic (University of Brasilia) and one public (EC2 Amazon), were created in order to study BioNimbus when applied to a simple workflow with real data.

A prototype of BioNimbus containing all the main controller services was implemented: *monitoring and scheduling service*, *discovery service* and a simple *storage service*, using an open

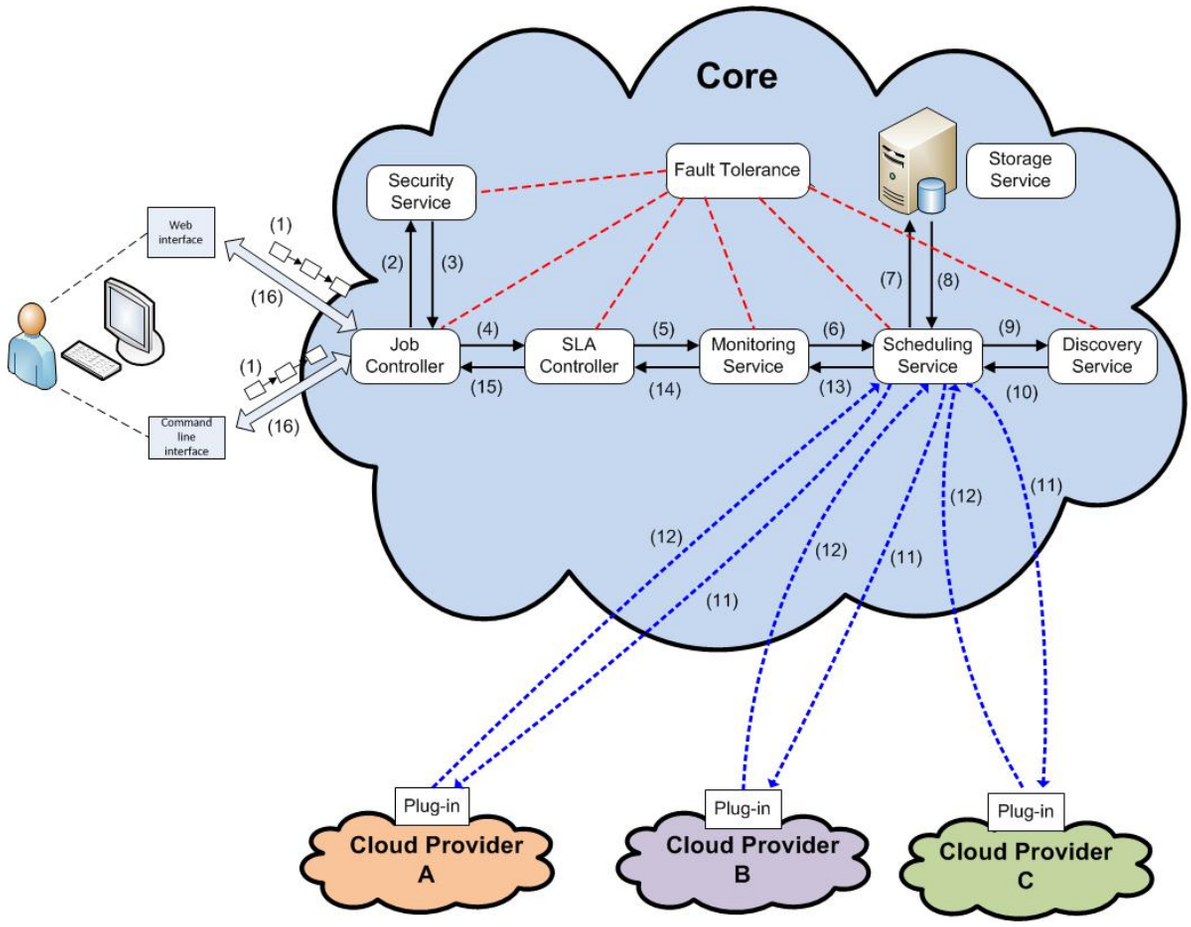

Figure 2. Example of a job execution in the BioNimbus hybrid federated cloud.

source implementation of the Zab protocol [59], which allows a distributed consensus among a group of processes. We also implemented Hadoop infrastructure plug-ins. Each plug-in provides information about the current status of its respective infrastructure, like number of cores, processing and storage resources and bioinformatics tools that can be executed in BioNimbus, as well as information of input and output files. The interaction of the user and the platform was implemented by a command line that sends requests. Services and plug-ins communicate through a P2P network based on the Chord protocol [67] .

In order to study the runtime performance of a workflow involving real biological data, we created a three-phase workflow in BioNimbus. The objective was to compare the time of a workflow running in a federated cloud to a single cloud.

4.1. Cloud providers

At the University of Brasilia, a Hadoop cluster was implemented with 3 machines, each one with two Intel Core 2 Duo 2.66Ghz (so a total of 6 cores), 4 GB RAM and 565 GB of storage. The Hadoop cluster executed Bowtie [44] with the Hadoop MapReduce (Hadoop streaming), with storage implemented with the Hadoop Distributed FileSystem (HDFS).

In addition, at Amazon EC2, a Hadoop cluster Cluster Hadoop was implemented with 4 virtualized machines, each one with two Intel Xeon 2.2 7Ghz (so a total of 8 cores), 8 GB RAM, and 1.6 TB of storage. The cluster also executed Bowtie.

Two Perl scripts implementing the workflow (SAM2BED and genome2interval) and the coverageBed program (integrating the BEDTolls suite [54]) were installed in each cloud provider.

4.2. Workflow, tools and data

Workflow

Now we describe the workflow used as our case study. The objective of the workflow was to identify differentially expressed genes in human kidney and liver cancerous cells [47, 60], with fragments of genes sequenced with Illumina technology [35]. The workflow consists of four phases (Figure 3): (i) mapping the input sequences onto the 24 human chromosome sequences; (ii) converting format from SAM (Bowtie) to BED (a specific format of the CoverageBED program); (iii) generating fixed intervals for all chromossomes based on their length, since this is the input for the CoverageBED program; and (iv) executing the CoverageBED program, which generates histograms showing the number of mappings for each interval.

The *mapping phase* has the objective of identifying the region of a reference genome where each input sequence was located. A set of sequences mapping in the same region allows the inferences that these sequences have the same structural organization of the reference genome.

The *CoverageBED* program [54] allowed the study of the expression level of the cancerous genes using histograms of the mapped input sequences onto the human reference genome, so that differentially expressed genes between kidney and liver cancer genes could be identified.

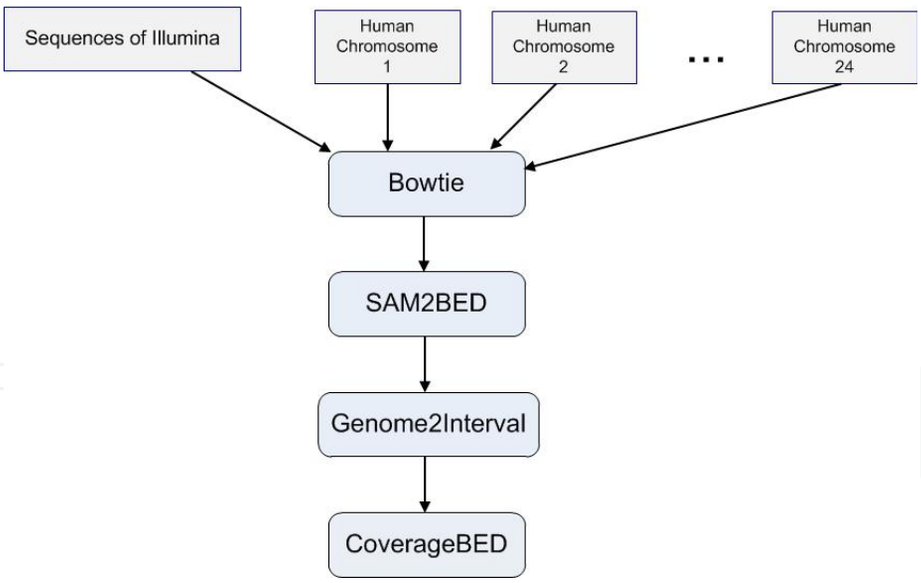

Figure 3. The workflow investigated the expression level of liver and kidney genes (generated by Illumina) mapping them to 24 human chromosome sequences.

Tools

Four tools were implemented in each cloud provider: Bowtie [44], SAM2BED (a Perl script), genome2interval (another Perl script) and coverageBed [54].

Data

The 24 human chromosome sequences were downloaded from HG19:

ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/Assembled_chromosomes/seq/

The reference site is:

http://www.ncbi.nlm.nih.gov/genome/assembly/293148/

Finally, the names of the files followed the format:

hs_ref_GRCh37.p5_chr*.fa.gz.

4.3. Implementation details

A message module allowed the communication among the services, having been created using the Nettycommunication library [36], which is responsible for the TCP connection event manager. Messages were serialized using both JSON format [23] and Jackson library [21], and file transfer was accomplished through the HTTP protocol `GET` and `PUT` methods. Message and file communications were realized using an unique TCP port, which avoided the necessity to create complex firewall rules. Besides, the message module is capable of multiplexing both message and file traffic. A simplified version of the *Chord* [67] protocol was implemented for the P2P network and plug-ins. We developed plug-in prototypes for Apache Hadoop and SunGridEngine. Java was the language used to implement the BioNimbus prototype.

Next, we briefly describe somefeatures of the services implemented on our BioNimbus prototype (Figure 4):

- Discovery service: this implementation used two execution threads. The first one is responsible for updating and cleaning the data structure storing information about the cloud providers. The second thread waits for P2P network messages that have to be treated by the discovery service. A data structure **map** was used for storing information about each federated cloud provider using a unique identifier. Besides, each cloud has a *timestamp* for its last mapping. To update the infrastructure, the first thread is executed in intervals of 30 seconds in order to send messages to all the BioNimbus members. The response of each plug-in is treated by the second thread, which updates the mapping with the received new information and corresponding *timestamp* for each execution. The first thread removes from the **map** those pieces of information that did not have their date modified in the last 90 seconds, which indicates that those cloud providers left the federation. The second thread also treats the requisition about the federation clouds, using the **map** maintained by the discovery service.
- Monitoring and scheduling service: to realize the work of receiving, monitoring and scheduling user jobs, three main data structures of type **map** were used. The first one, called `PendingJobs`, maps each job identifier to its information and also represents those jobs waiting to be scheduled. The second one, named `RunningJobs`, maps each executing task identifier to its information and the job to which it belongs. The third data structure, called `CancelingJobs`, maps the task identifier to its corresponding job and to the user requiring its cancelation.

  In the monitoring service, there is a thread responsible for waiting the user requests and responses received from other services of the infrastructure. When a request initiates a

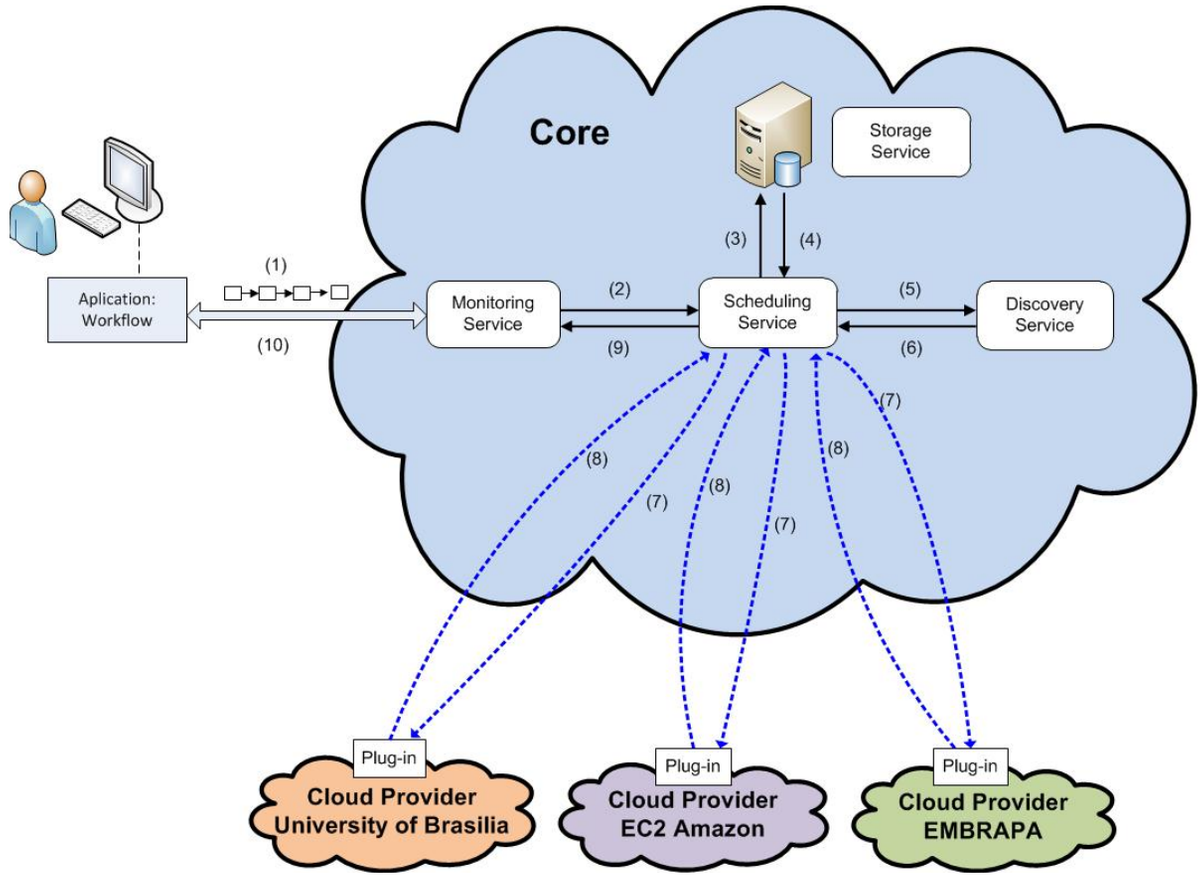

Figure 4. The services implemented for our case study, noting that we used two cloud providers in the BioNimbus prototype.

job (`JobStartReq`) is received, this thread generates an unique identifier for this job and saves this informaticon in the `PendingJobs` map. Next, it calls the scheduling policy, which returns a mapping among the jobs and the plug-ins that can execute them. Thus, when the jobs are all scheduled, this thread sends requests in order to create tasks (`TaskStartReq`) that have to be executed in the cloud providers, and waits for their corresponding outputs (`TaskStartReply`). When an output is received, the service removes the job from the `PendingJobs` map and creates an input in the `RunningJobs` map, with information about the job and its corresponding tasks, removing a job when all its tasks finish. As previouly mentioned, a new *DynamicAHP* algorithm was implemented in BioNimbus [12], which is based on on a decision making strategy proposed by [61].

Another thread in the monitoring service, executed at intervals of 15 seconds, is responsible for following the jobs. First, it sends status requests (`TaskStatusReq`) to each job registered in `RunningJobs`. The response (`TaskStatusReply`), treated by the previous described thread, can again initiate the scheduling service according to some parameters. Cancelling messages (`TaskCancelReq` and `TaskCancelReply`) will be sent, and the job will be reinserted in the `PendingJobs` map and removed from the `RunningJobs`. This thread also verifies whether there are pending jobs in `PendingJobs`, initiating another scheduling process in this case, and sending query messages to the discovery service (`CloudReq`) and to the storage service (`ListReq`), whose responses (`CloudReply` and `ListReply`) will be received by the first thread and used by the scheduling policy when needed.

- Storage service: two threads were used for its implementation. The first one waits for the requests sent by other services. To treat the request of saving files (`StoreReq`), the storage service executes the storage policy adopted in BioNimbus. For this case study, we used a method based on a round-robin of the plug-ins that informed having enough space to store the file. When a cloud is chosen, a response (`StoreReply`) is sent to the service making the request, which will send the file to the cloud indicated by the storage service. When this transfer finishes, the plug-in receiving the file storage sends a special message (`StoreAck`), which contains information that will keep correct the federation file table.
  In the case study, a simple backend was implemented to maintain the federation file table. Every time a new confirmation is received by the storage service, it adds an input in the map file with the file identifier containing information such as name, size and storage cloud. This mapping is stored in JSON format [23] in a file in the federation file system of the cloud where the service will be executed. When initiating its execution, the storage service verifies if the map file left and load in memory the federation file table last status.
  The other two types treated by the first thread are file list (`ListReq`) and localization (`GetReq`). For the first case, the thread builds a response (`ListReply`) with the mapping loaded in memory. For localization, it builds a response (`GetReply`) searching for the cloud information in the map using the request identifier.
  Finally, another thread is executed at intervals of 30 seconds requesting to the discovery service the current configuration of the federation (`CloudReq` message). The received information is used by the storage policy.

4.4. Results

We executed the workflow at the University of Brasilia and the Amazon EC2, and on both cloud providers (Table 1).

Cloud Providers	hour:minute:second
University of Brasilia (UnB)	1:11:47
Amazon EC2	1:18:44
Both clouds (UnB and EC2)	1:09:07

Table 1. Workflow execution time on each cloud and on the BioNimbus federated cloud.

We measured how the file transfer time affected the job execution total time. Table 2 and Figure 5 shows the total and file transfer times of the 18 longest jobs of the workflow, as well as the percentage of the file transfer time related to the total time. These percentages show that file transfer represents at least 50% of the total time of this job execution. This means that in federated clouds executing data-driven bioinformatics applications, storage services have to be especially designed to minimize as much as possible huge file transfers.

We also investigated how the time execution of a job was affected when sent to execution in a cloud provider, taking a long time, being cancelled and returning to the list of pending jobs to be executed again. There were seven jobs cancelled, the first seven jobs in Table 2 with the longest times to be executed. When making an experiment without cancelling jobs, we obtained greater times, when compared to the experiment with cancelling since they were sent to clouds almost idle.

We mention now some points that can affect BioNimbus performance: (i) the scheduler does not consider jobs being transferred and identifies CPUs involved in these transfers as idle; (ii)

Total time (seconds)	File transfer time (seconds)	Percentage of file transfer time related to the total time
4230.297	2114.996	50.0%
4123.492	2264.552	54.9%
4098.571	2337.454	57.0%
4030.492	2297.580	57.0%
3807.501	2229.992	58.6%
3145.645	2168.201	68.9%
3113.729	2116.199	68.0%
3066.488	2058.771	67.1%
3032.701	2018.942	66.6%
3001.165	2137.157	71.2%
2952.875	2087.761	70.7%
2849.506	2074.117	72.8%
2801.489	2023.309	72.2%
2680.382	1892.002	70.6%
2587.076	2006.842	77.6%
2579.184	1959.727	76.0%
2533.254	1928.888	76.1%
2405.470	1899.626	79.0%

Table 2. Total and file transfer times of the longest jobs executed in BioNimbus.

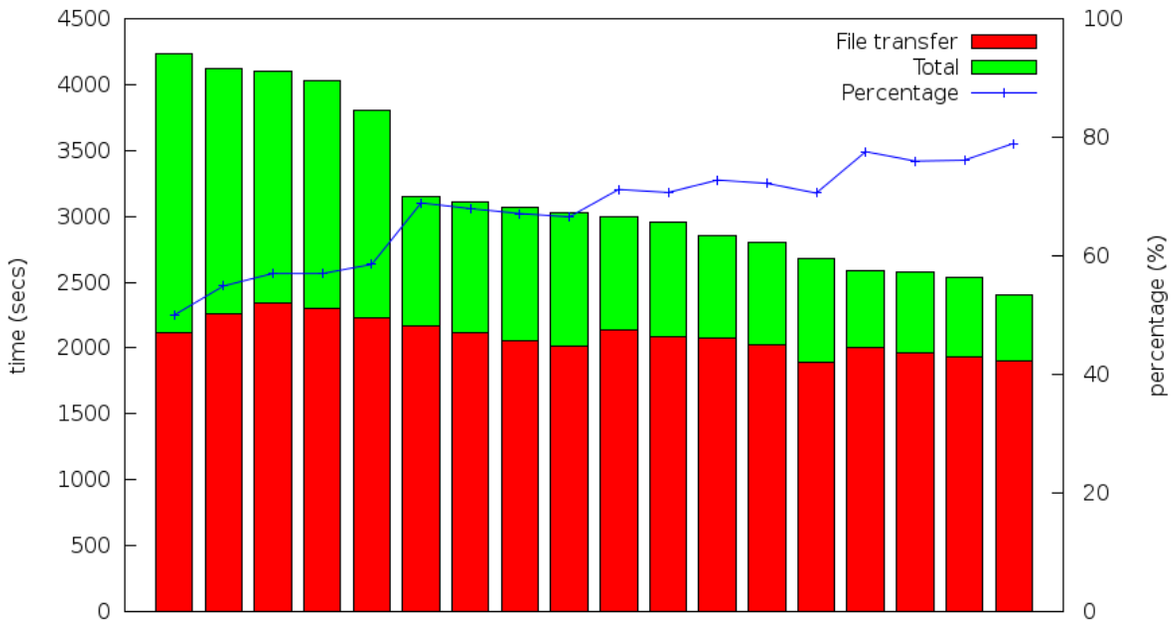

Figure 5. Comparing the total and file transfer times of the longest jobs executed in BioNimbus. The file transfer time is colored red, while its percentage related to the total time is shown in blue.

the input files are all simultaneously downloaded, i.e. there are no priorities for downloads; (iii) jobs are now canceled based only on the wait time in the pending jobs list, i.e. the file transfering time is not considered; and (iv) jobs with small input files that were sent to a cloud provider after jobs with large input files got executed earlier, while the later were still downloading their input data.

Table 3 and Figure 6 show the number of jobs executed in a single cloud provider and on both. Note that, including the transfer time, jobs with smaller inputs execute faster on two cloud providers, since the possibility to cancel delayed jobsthat are running and scheduling them again lowered the total execution time. Besides, when files are small, the time to transfer files is rapid, while when they are large the transfer time strongly affects the total execution time (as shown in Table 2). Thus, for large files, the storage policy has to be very carefully designed using replication and fragmentation in order to significantly decrease file transfer time.

Cloud Providers	until 200 seconds	between 200 seconds and 1000 seconds	above 1000 seconds
University of Brasilia	34	30	32
EC2 Amazon	37	27	32
UnB and EC2	64	8	24

Table 3. Number of executed jobs, where time includes the file transfer time.

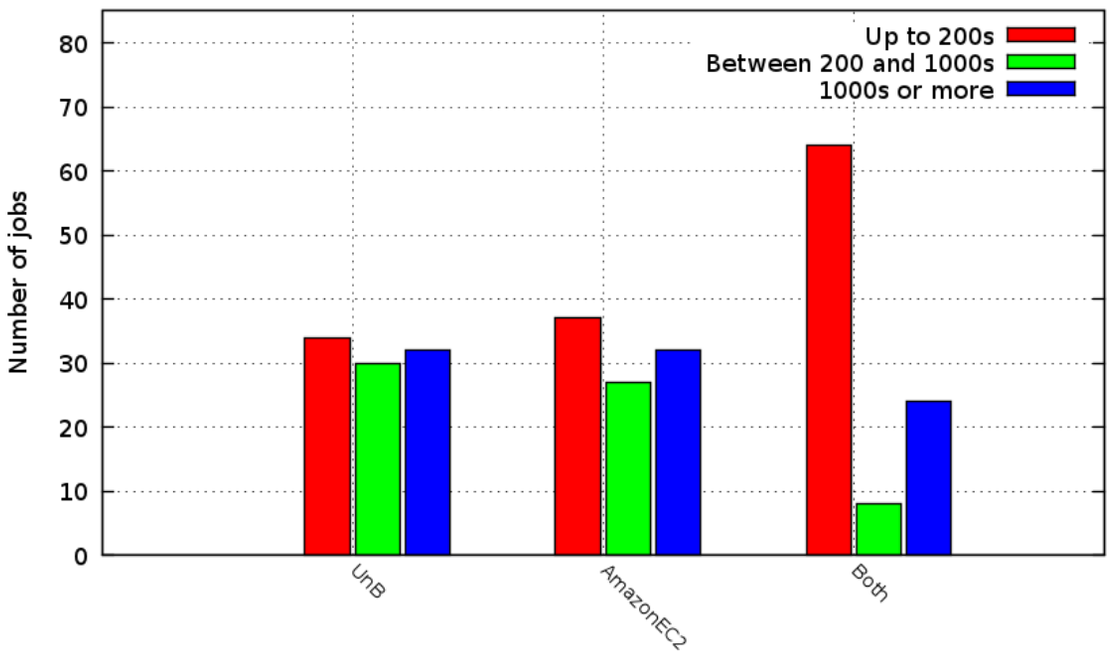

Figure 6. Comparing the number of executed jobs in BioNimbus, where time (in seconds) includes the file transfer time.

5. Related work

In this section, we discuss cloud projects designed to accelerate execution and increase the amount of storage available to bioinformatics applications. When compared to BioNimbus, these projects are dedicated to particular applications or are executed in a single cloud environment. BioNimbus intends to integrate public and private centers offering bioinformatics applications in one single platform using the hybrid federation cloud paradigm.

Cloudburst [64] parallel algorithm is optimized for mapping DNA fragments, also known as short read sequences (SRSs), to a reference genome. The execution time varies almost linearly with the increase in the number of processors. The mapping of millions of SRSs to the human genome, executed in 24 cores, is thirty times faster when compared to other non-distributed applications [44, 65]. CloudBurst uses the MapReduce model.

Crossbow [43] is a pipeline developed in the infrastructure provided by the Apache Hadoop streaming mode. It combines the Bowtie [43] SRS mapping tool, performed during the map phase, with the SOAPsnp [46] tool to identify SNPs, processed during the reduce phase. During the execution of the workflow, the SRSs are sent as input to the nodes of the Hadoop cluster, which executes the map phase. In this phase, the SRSs are mapped to a reference genome using Bowtie. Afterwards, the mappings are joined with parts of the reference genome, and each group is sent to a node that executes the reduce phase. The SOAPsnp tool is used to detect SNPs in the already analyzed parts of the genome. The execution time for about 2.6 billion SRSs and the entire human genome used as a reference took a little more than 3 hours in a 320 core cluster of the Amazon EC2 [2] infrastructure. The experiments cost less than US$ 100.

Myrna [42] identifies differentially expressed genes in large sets of sequenced data. The workflow combines a mapping phase with a statistical analysis phase, performed with R [55], which is able to analyze more than one billion SRSs in a little more than 90 minutes, using 320 cores and costing around US$ 75.

The RSD (Reciprocal Smallest Distance) comparative genomics algorithm, composed of different bioinformatics tools, was adapted to be executed in the Amazon EC2 infrastructure, having obtained expressive results [72].

[3] created the Cloud Virtual Resource (CloVR), a desktop application for automated sequence analysis using cloud computing resources. CloVR is implemented as a single portable virtual machine (VM) that provides several automated analysis pipelines for microbial genomics, whole genome and metagenome sequence analysis. The CloVR VM runs on a personal computer, uses local computer resources and addresses problems arising in constructing bioinformatics workflows.

[4] noted that genomic applications are limited by the "bioinformatics bottleneck", due to computational costs and infrastructure needed to analyze the enormous amounts of SRSs. They presented benchmark costs and runtimes for microbial genomics applications, microbial sequence assembly and annotation, metagenomics and large-scale BLAST. They also analyzed workflows (also called pipelines) implemented in the CloVR virtual machine running in Amazon EC2, having achieved cost-efficient bioinformatics processing using clouds, and thereby claiming that this is an interesting alternative to local computing centers.

[53] adapted a particular peptide search engine called X!Tandem to Hadoop MapReduce. Their MR-Tandem application runs on any Hadoop cluster, but it was especially designed to run on Amazon Web Services. They modified the X!Tandem C++ program and created a Python script for driving Hadoop clusters, which includes the Amazon Web Services (AWS) Elastic Map Reduce (EMR) used by the modified X!Tandem as a Hadoop streaming mapper and reducer.

[75] worked on pathway-based or gene set analysis of expression data, having developed a gene set analysis algorithm for biomarker identification in a cloud. Their YunBe tool is ready

to use on the Amazon Web Services. YunBe performed well when compared to desktop and cluster executions. YunBe is open-source and freely accessible within the Amazon Elastic MapReduce service.

[27] ported two bioinformatics applications, a pairwise Alu sequence alignment application and an Expressed Sequence Tag (EST) sequence assembly program, to the cloud technologies Apache Hadoop and Microsoft DryadLINQ. They studied the performance of both applications in these two cloud technologies, comparing them with traditional MPI implementation. They also analyzed how non-homogeneous data affected the scheduling mechanisms of the cloud technologies, and compared performance of the cloud technologies under virtual and nonvirtual hardware platforms.

[32] used cloud computing for scientific workflows, and discussed a case study of a widely used astronomy application.

The Bio-Cloud Computing platform [9] was designed to support large-scale bioinformatics processing. It has five main bio-cloud computing centers, with a total peak performance up to 157 Teraflops, 33.3 TB memory and 12.6 PB storage.

Recently, many bioinformatics applications have been ported to clouds [33, 37, 40], noting that they offer user-friendly web interfaces and efficiency in the execution of tools that extensively use memory and storage resources.

6. Conclusion and future work

In this work, we proposed a hybrid federated cloud computing platform called BioNimbus, which aims at integrating and controlling different bioinformatics tools in a distributed, transparent, flexible and fault tolerant manner, also providing highly distributed processing and large storage capability. The objective was to make possible the use of tools and services provided by multiple institutions, public or private, that could be easily aggregated to the federated cloud. We also discussed a case study in a prototype of BioNimbys including two cloud providers, in order to verify its performance in practice. We created a bioinformatics workflow for identifying liver and kidney cancerous differentially expressed genes, and measured its total time execution on each single cloud provider and on all of them.

The next step is to study different scheduling strategies for the *scheduling service*, in order to improve its efficiency when choosing a cloud provider to execute jobs. Our results showed that the execution time is strongly affected by the file transfer time, implying that we have to carefully design the *storage service*; we plan to use data replication and fragmentation to address this problem. A *fault tolerance service* to check the cloud providers and other services status will be developed and evaluated. We also plan to use an adaptive fault monitoring algorithm, as proposed by [18, 30] and [70], which are more adaptable to be used in a large-scale distributed environment. It is also important to include a *security service* and an *SLA service* in the federated platform. Finally, we will investigate the use of a Workflow Management System (WfMS) in BioNimbus.

Acknowledgments

M.E.M.T.Walter would like to thank to FINEP (Project number 01.08.0166.00) and all the authors would like to thank Daniel Saad for having written the Perl scripts for the workflow.

Author details

Hugo Saldanha, Edward Ribeiro, Carlos Borges, Aletéia Araújo, Ricardo Gallon, Maristela Holanda and Maria Emília Walter
University of Brasília, Brasil

Roberto Togawa
Embrapa/Genetic Resources and Biotechnology, Brasil

João Carlos Setubal
University of São Paulo, Brasil

7. References

[1] 454 [2012]. 454 life sciences. roche diagnostics corporation, http://www.454.com/.

[2] Amazon [2012]. Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/.

[3] Angiuoli, S. V., Matalka, M., Gussman, A., Galens, K., Vangala, M., Riley, D. R., Arze, C. & White, J. R. [2011]. Clovr: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing, *BMC Bioinformatics* 12(356): 1–15.

[4] Angiuoli, S. V., White, J. R., Matalka, M., White, O. & Fricke, W. F. [2011]. Resources and costs for microbial sequence analysis evaluated using virtual machines and cloud computing, *PLoS ONE* 6(10): e26624.

[5] Apache [2011]. The Apache Software Foundation: Apache, http://couchdb.apache.org/.

[6] Apache [2012]. Apache Hadoop, http://hadoop.apache.org/.

[7] Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R. & Stoica, I. [2003]. Looking up data in p2p systems, *Commun. ACM* 46(2): 43–48.
URL: *http://doi.acm.org/10.1145/606272.606299*

[8] Bermbach, D., Klems, M., Tai, S. & Menzel, M. [2011]. Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs, *IEEE International Conference on Cloud Computing (CLOUD)*, IEEE, pp. 452–459.

[9] BGI [2012]. Bio-Cloud Computing, http:www.genomics.cn/en/navigation/show_navigation?nid=4143.

[10] Bhandari, A. & Singh, M. [2010]. 2010 the oauth 1.0 protocol, internet engineering task force (ietf), rfc:5849.
URL: *http://tools.ietf.org/html/rfc5849*

[11] Bittman, T. J. [2008]. Th evolution of the cloud computing market, http://blogs.gartner.com/thomas_bittman/2008/11/03/the-evolution-of-the-cloud-computing-market.

[12] Borges, C. A. L., Saldanha, H. V., Ribeiro, E., Holanda, M. T., Araujo, A. P. F. & Walter, M. E. M. T. [2012]. Task scheduling in a federated cloud infrastructure for bioinformatics applications, *in* INSTICC (ed.), *2nd International Conference on Cloud Computing and Services Science ()*, CLOSER 2012, Porto, Portugal, pp. 1–7.

[13] Borthakur, D. [2008]. The Apache Software Foundation: HDFS Architecture, http://hadoop.apache.org/common/docs/r0.20.2/hdfs_design.pdf.

[14] Buyya, R., Ranjan, R. & Calheiros, R. N. [2010]. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services, *Proceedings of the*

10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2010), Springer, pp. 21–23.

[15] Celesti, A., Tusa, F., Villari, M. & Puliafito, A. [2008]. How to enhance cloud architectures to enable cross-federation, *3rd IEEE International Conference on Cloud Computing (IEEE Cloud 2010)*, Miami, Florida, USA, pp. 337–345.

[16] Chandra, T. D. & Toueg, S. [1996]. Unreliable failure detectors for reliable distributed systems, *Journal of the ACM* 43: 225–267.

[17] Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A. & Gruber, R. [2008]. Bigtable: A distributed storage system for structured data, *ACM Transactions on Computer Systems (TOCS)* 26(2): 1–26.

[18] Chen, W. [2000]. On the quality of service of failure detectors, *IEEE Transactions on Computers* 51: 561–580.

[19] Chodorow, K. & Dirolf, M. [2010]. *MongoDB: the definitive guide*, O'Reilly Media, Inc.

[20] Clayman, S., Galis, A., Chapman, C., Toffetti, G., Rodero-Merino, L., Vaquero, L. M., Nagin, K. & Rochwerger, B. [2010]. Monitoring Service Clouds in the Future Internet, *Towards the Future Internet - Emerging Trends from European Research*, IOS Press.

[21] Codehaus [2012]. Jackson Java JSON-Processor, `http://jackson.codehaus.org`.

[22] Cooper, B., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H., Puz, N., Weaver, D. & Yerneni, R. [2008]. PNUTS: Yahoo!'s hosted data serving platform, *Proceedings of the VLDB Endowment*, VLDB Endowment, pp. 1277–1288.

[23] Crockford, D. [2006]. The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627 (Informational).
URL: *http://www.ietf.org/rfc/rfc4627.txt*

[24] da Silva, C. A. R. F. O. [2011]. *Data modeling with NoSQL: How, When and Why*, Master's thesis, University of Porto.

[25] Dean, J. & Ghemawat, S. [2004]. MapReduce: simplified data processing on large clusters, *6th Conference on Symposium on Operating Systems Design & Implementation*, USENIX Association, Berkeley, CA, EUA, pp. 10–10.

[26] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. & Vogels, W. [2007]. Dynamo: amazon's highly available key-value store, *SIGOPS Oper. Syst. Rev.* 41(6): 205–220.
URL: *http://doi.acm.org/10.1145/1323293.1294281*

[27] Ekanayake, J., Gunarathene, T. & Qiu, J. [2011]. Cloud technologies for bioinformatics applications, *IEEE Transactions on Parallel and Distributed Systems* 22(6): 998–1011.

[28] Elmroth, E. & Larsson, L. [2009]. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds, *Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing*, GCC '09, IEEE Computer Society, Washington, DC, USA, pp. 253–260.
URL: *http://dx.doi.org/10.1109/GCC.2009.36*

[29] Foster, I., Zhao, Y., Raicu, I. & Lu, S. [2008]. Cloud Computing and Grid Computing 360-Degree Compared, *Grid Computing Environments Workshop GCE '08*, pp. 1–10.

[30] Hayashibara, N., Défago, X., & Rami Yared, T. K. [2004]. The phi Accrual Failure Detector, *23rd IEEE International Symposium on Reliable Distributed Systems*, pp. 66–78.

[31] Hayashibara, N., Défago, X., Yared, R. & Katayama, T. [2004]. The ϕ accrual failure detector, *RR IS-RR-2004-010, Japan Advanced Institute of Science and Technology*, pp. 1–16.

[32] Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K. a. B. B. & Good, J. [2008]. On the use of cloud computing for scientific workflows, *3rd International Workshop on Scientific Workflows and Business Workflows Standards in e-Science*, SWBES 2008, IEEE Digital Library, pp. 640–645.

[33] Hong, D., Rhie, A., Park, S.-S., Lee, J., Ju, Y. S., Kim, S., Yu, S.-B., Bleazard, T., Park, H.-S., Rhee, H., Chong, H., Yang, K.-S., Lee, Y.-S., Kim, I.-H., Lee, J. S., Kim, J.-I. & Seo, J.-S. [2012]. FX: an RNA-Seq analysis tool on the cloud, *Bioinformatics* 28(5): 721–723.
URL: *http://dx.doi.org/10.1093/bioinformatics/bts023*

[34] Hunt, P., Konar, M., Junqueira, F. P. & Reed, B. [2010]. Zookeeper: wait-free coordination for internet-scale systems, *USENIX conference on USENIX annual technical conference*, USENIXATC'10, USENIX Association, pp. 11–11.

[35] Illumina [2012]. Illumina life sciences, `http://www.illumina.com`.

[36] Inc., R. H. [2012]. Netty - the Java NIO Client Server Socket Framework, `http://www.jboss.org/netty`.

[37] Jourdren, L., Bernard, M., Dillies, M.-A. A. & Le Crom, S. [2012]. Eoulsan: A Cloud Computing-Based Framework Facilitating High Throughput Sequencing Analyses., *Bioinformatics (Oxford, England)* .
URL: *http://dx.doi.org/10.1093/bioinformatics/bts165*

[38] Khetrapal, A. & Ganesh, V. [2006]. Hbase and hypertable for large scale distributed storage systems, Dept. of Computer Science, Purdue University, http://www.uavindia.com/ankur/downloads/HypertableHBaseEval2.pdf.

[39] Kossmann, D., Kraska, T., Loesing, S., Merkli, S., Mittal, R. & Pfaffhauser, F. [2010]. Cloudy: a modular cloud storage system, *Proceedings of the VLDB Endowment*, VLDB Endowment, pp. 1533–1536.

[40] Krampis, K., Booth, T., Chapman, B., Tiwari, B., Bicak, M., Field, D. & Nelson, K. [2012]. Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community, *BMC Bioinformatics* 13(1): 42+.
URL: *http://dx.doi.org/10.1186/1471-2105-13-42*

[41] Lakshman, A. & Malik, P. [2010]. Cassandra: a decentralized structured storage system, *SIGOPS Oper. Syst. Rev.* 44(2): 35–40.
URL: *http://doi.acm.org/10.1145/1773912.1773922*

[42] Langmead, B., Hansen, K. & Leek, J. [2010]. Cloud-scale RNA-sequencing differential expression analysis with Myrna, *Genome Biology* 11(8): R83.

[43] Langmead, B., Schatz, M. C., Lin, J., Pop, M. & Salzberg, S. [2009]. Searching for SNPs with cloud computing, *Genome Biology* 10(11): R134.

[44] Langmead, B., Trapnell, C., Pop, M. & Salzberg, S. [2009]. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, *Genome Biology* 10(3): R25.

[45] Larrea, M., Fernandez, A., Arevalo, S., Carlos, J. & Carlos, J. [2002]. Eventually consistent failure detectors, *Brief Announcement, 14th International Symposium on Distributed Computing (DISC'2000)*, pp. 326–327.

[46] Li, R., Li, Y., Fang, X., Yang, H., Wang, J., Kristiansen, K. & Wang, J. [2009]. SNP detection for massively parallel whole-genome resequencing, *Genome Research* 19(6): 1124–1132.
URL: *http://dx.doi.org/10.1101/gr.088013.108*

[47] Marioni, J. C., Mason, C. E., Mane, S. M., Stephens, M. & Gilad, Y. [2008]. Rna-seq: An assessment of technical reproducibility and comparison with gene expression arrays,

Genome Research 18: 1509–1517.
URL: *http://genome.cshlp.org/cgi/content/full/18/9/1509*

[48] Mell, P. & Grance, T. [2009]. The NIST Definition of Cloud Computing, *National Institute of Standards and Technology* 53(6): 50.
URL: *http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc*

[49] Oppenheimer, D., Albrecht, J., Patterson, D. & Vahdat, A. [2004]. Scalable wide-area resource discovery, *Technical Report UCB/CSD-04-1334*, EECS Department, University of California, Berkeley.
URL: *http://www.eecs.berkeley.edu/Pubs/TechRpts/2004/5465.html*

[50] Orend, K. [2010]. *Analysis and classification of nosql databases and evaluation of their ability to replace an object-relational persistence layer*, Master's thesis, Fakultät für Infoormatik, Technische Universität München.

[51] Padhy, R., Patra, M. & Satapathy, S. [2011]. Rdbms to nosql: Reviewing some next-generation non-relational databases, *International Journal of Advanced Engineering Science and Technologies* 11(1): 15–30.

[52] Pashalidis, A. & Mitchell, C. J. [2003]. A taxonomy of single sign-on systems, *Information Security and Privacy, 8th Australasian Conference, ACISP 2003*, Springer-Verlag, pp. 249–264.

[53] Pratt, B., Howbert, J. J., Tasman, N. & Nilsson, E. J. [2011]. MR-Tandem: Parallel X!Tandem using Hadoop MapReduce on Amazon Web Services, *Bioinformatics* 8: 1–12.

[54] Quinlan, A. R. & Hall, I. M. [2010]. BEDTools: a flexible suite of utilities for comparing genomic features, *Bioinformatics* 26(6): 841–842.
URL: *http://bioinformatics.oxfordjournals.org/content/26/6/841.abstract*

[55] R Development Core Team [2011]. *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.
URL: *http://www.R-project.org*

[56] Ranjan, R., Chan, L., Harwood, A., Karunasekera, S. & Buyya, R. [2007]. Decentralised resource discovery service for large scale federated grids, *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '07, IEEE Computer Society, Washington, DC, USA, pp. 379–387.
URL: *http://dx.doi.org/10.1109/E-SCIENCE.2007.27*

[57] Recordon, D. & Reed, D. [2006]. Openid 2.0: a platform for user-centric identity management, *Proceedings of the second ACM workshop on Digital identity management*, DIM '06, ACM, New York, NY, USA, pp. 11–16.

[58] Redkar, T. [2011]. *Windows Azure Platform*, Vol. 1, 2th edn, Apress, Berkeley, CA, USA.

[59] Reed, B. & Junqueira, F. P. [2008]. A simple totally ordered broadcast protocol, *2nd Workshop on Large-Scale Distributed Systems and Middleware*, LADIS'08, ACM, New York, NY, USA, pp. 2:1–2:6.

[60] Robinson, M. & Oshlack, A. [2010]. A scaling normalization method for differential expression analysis of RNA-seq data, *Genome Biology* 11(3): R25+.
URL: *http://dx.doi.org/10.1186/gb-2010-11-3-r25*

[61] Saaty, T. L. [1990]. How to make a decision: The analytic hierarchy process, *European Journal of Operational Research* 48(1): 9 – 26.

[62] Saldanha, H. V., Ribeiro, E., Holanda, M., Araujo, A., Rodrigues, G., Walter, M. E. M. T., Setubal, J. C. & Davila, A. [2011]. A cloud architecture for bioinformatics workflows,

in L. INSTICC (ed.), *1st International Conference on Cloud Computing and Services Science*, CLOSER 2011, pp. 1–8.
[63] Sanderson, D. [2009]. *Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure*, 1st edn, O'Reilly Media, Inc.
[64] Schatz, M. C. [2009]. CloudBurst: Highly Sensitive Read Mapping with MapReduce, *Bioinformatics* 25: 1363–1369.
[65] Smith, A., Xuan, Z. & Zhang, M. [2008]. Using quality scores and longer reads improves accuracy of Solexa read mapping, *BMC Bioinformatics* 9(1): 128.
[66] Solid [2012]. Life technologies & applied biosystems, `http://www.appliedbiosystems.com/`.
[67] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F. & Balakrishnan, H. [2003]. Chord: a scalable peer-to-peer lookup protocol for internet applications, *IEEE/ACM Trans. Netw.* 11(1): 17–32.
[68] Tanenbaum, A. S. [2002]. *Computer Networks (International Edition)*, fourth edn, Prentice Hall.
[69] Tolone, W., Ahn, G.-J., Pai, T. & Hong, S.-P. [2005]. Access control in collaborative systems, *ACM Comput. Surv.* 37(1): 29–41.
[70] van Renesse, R., Minsky, Y. & Hayden, M. [1998]. A gossip-style failure detection service, *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Middleware '98, Springer-Verlag, London, UK, pp. 55–70.
[71] Vaquero, L. M., Rodero-Merino, L., Caceres, J. & Lindner, M. [2008]. A break in the clouds: towards a cloud definition, *SIGCOMM Comput. Commun. Rev.* 39: 50–55.
[72] Wall, D., Kudtarkar, P., Fusaro, V., Pivovarov, R., Patil, P. & Tonellato, P. [2010]. Cloud computing for comparative genomics, *BMC Bioinformatics* 11(1): 259.
[73] Wang, J., Varman, P. & Xie, C. [2011]. Optimizing storage performance in public cloud platforms, *Journal of Zhejiang University-Science C* 12(12): 951–964.
[74] Wu, L. & Buyya, R. [2010]. Service level agreement (sla) in utility computing systems, *CoRR* abs/1010.2881.
[75] Zhang, L., Gu, S., Wan, B., Liu, Y. & Azuaje, F. [2011]. Gene set analysis in the cloud, *Bioinformatics* 13: 1–10.