Academic Year 2015/2016

UNIVERSITY "FEDERICO II" – NAPLES

PhD In Computational biology and Bioinformatics

XXVIII CYCLE

A DISTRIBUTED ENVIRONMENT FOR HIGH THROUGHPUT SEQUENCING AND OTHER BIOINFORMATIC DATA ANALYSIS

Coordinator:

Prof. Sergio Cocozza

Student:

Mario Zanfardino

Tutor:

Prof. Giovanni Paolella

FOREWORD

I wish to express my gratitude and appreciation to the many persons whose ideas and help have strongly contributed to the development of this work. I am really grateful to my tutor, professor Giovanni Paolella, who has followed the development of my studies and work with continuous attention, supporting my growth with precious suggestions and contributions. In particular, of all the colleagues, I wish to thank Angelo Boccia, who has carefully followed my progression during these years. He has been a good colleague, whose support and encouragement have been very important for me, during the entire PhD period. I would like to thank all the colleagues that shared with me the laboratory life during these years. Finally I wish to thank my family and especially my girlfriend Luciana, who have supported me since the beginning, with encouragement, confidence and enthusiasm.

1. INTRODUCTION

Current experimental procedures in life science and biotechnology produce huge data volumes (an example is the growth, in recent years, of completed genome sequencing, Fig.1) and require, as a consequence, increasingly complex analysis tools as well as important hardware resources. An important cause of the exponential increase in biological data production is related to improvements in sequence technologies and, in particular in high-throughput sequencing. Compared to previous technologies such as Sanger sequencing, current approaches, based on novel sequence determination procedures implemented within a number a platforms such as Illumina, Roche 454, Ion Torrent and SOLiD, allow fast DNA sequencing much more rapidly and at lower costs. Analysis of this exponentially increasing amount of genomic, trascriptomic and proteomic sequence data, and related information, represents a challenge for biomedical research, with requirements for computer storage and processing power that easily match and exceed the current growth of available data processing power. Programs can also be developed through faster algorithms that reduce the request of computational power for the most time consuming tasks. Complex bioinformatic analyses require advanced programming and database skills and the user is typically required to install, configure and update various software tools and databases. Even though sometimes scientists with biological background and laboratory experience have these skills, bioinformatic programs may have hardware requirements, which cannot be satisfied by personal computers. In these cases, access to remotely installed tools becomes typically essential and different modes of access and execution are needed. The vast majority of applications in the bioinformatics field are available through command line interfaces, that make it possible to indifferently access local and remote applications. Today, many bioinformatic tools are available via a web-browser interface which facilitates their use also to the unsophisticated users. In both cases, the need to transfer large amounts of data, limits the advantage offered by remotely accessible tools and the search for more efficient access methods is important, especially for large-scale biological data analysis.

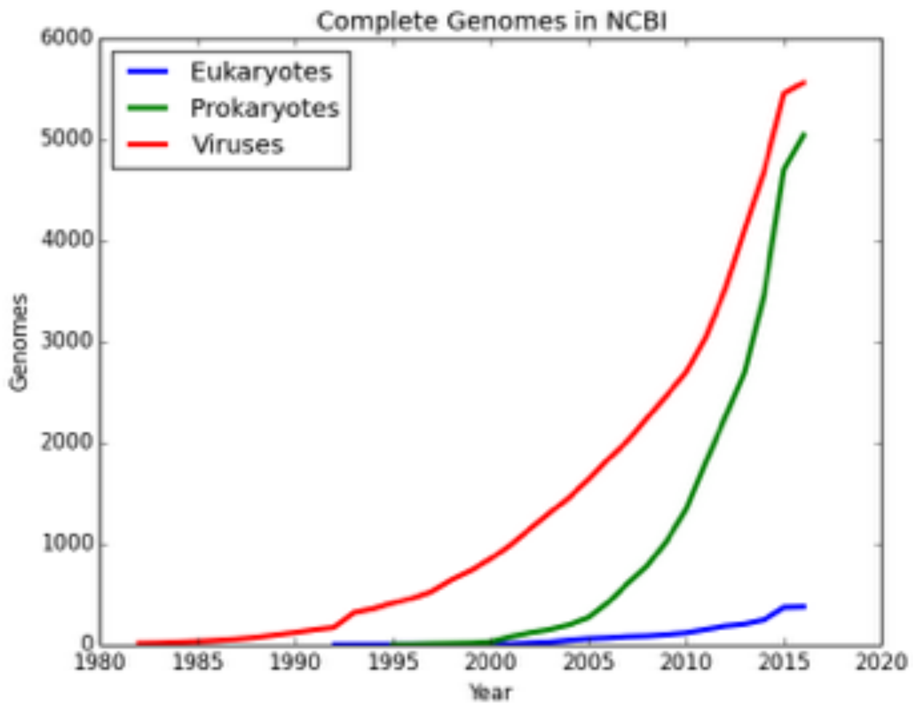


Figure 1. Growth of completed genome sequencing for Eukaryotes, Prokaryotes and Viruses. Data from NCBI Genome reports folder (ftp site).

1.1 Local data processing

A common scenario is based on local processing of personal data files and data downloaded from remote resources and. This requires tools for data analysis which can be made available by creating a local bioinformatics work environment, through downloading and installing the necessary programs and support databases. Most programs are available through command line interfaces, and have been developed by using different programming languages (C, Perl, Python, Java etc.) to run on different operating system. Commonly used operating systems are UNIX-based and provide powerful command-line tools that make scripting and performing automated analyses relatively easy. In this environment, processes can be automated by scripting and there is no need for data transfer. However, the number of available software tools results in increasing amount of time and resources spent in installing, configuring and maintaining software rather than in doing research work. To address these problems many solutions are available, some of which include locally shared servers based on customized Linux distributions, modified to fit the needs of the biomedical researchers community. Examples are BioLinux, Scibuntu, Open Discovery, BioSLAX, DNALinux, among others. However, the costs implied by maintaining such environments often outdo the capacity of local environment and anyway these solutions is restrictive for several reasons:

- in many cases, local computer power cannot satisfy the requests of some bioinformatic tools and analyses;
- programs are often developed to operate under Linux based environments, which may be not completely familiar for the user;
- some programs need large size databases or reference data such as entire genomes, implying capacity and maintainability, as well as, regular updating;
- many programs require additional support installation of specific libraries an software dependencies, which can increase the complexity of program installation;

1.2 Use of remotely installed programs

Many problems of local data processing can be overcome through the use of remotely installed programs. In this way a large variety of software tools for different types of data analyses may be used remotely, thus eliminating the need for local installation, software maintenance and updates, while providing up-to-date services for bioinformatic data analysis. There are a number of different ways to connect to machines that provide remotely installed programs, starting from the secure shell

(SSH) command line interfaces, which provide a secure remote connection between computers and has the advantage of being available on every Unix system.

1.2.1 Web interfaces

An alternative way to access remotely installed bioinformatics services is through WUI (Web User Interfaces) used as a front-end to programs. Being based on web interfaces, use of these services remains independent from local hardware and operating system. Examples of such user interfaces are PISE, Mobyle, wEMBOSS and GenePattern.

- PISE and Mobyle provide a web interface for bioinformatics applications originally developed to run through command-line and include methods for biological databases management as well as conversion of commonly used bioinformatic data formats. The use of Pise and its capacity to provide web page structures for bioinformatics programs, has advantages such as a graphic interface and the opportunity to know the available options without the need to access the manual. Based on a similar approach,, Mobyle provides a unified web-based framework through the integration of server installed programs with remotely available ones. This web-based framework offers the advantage of a flexible user interfaces with no costs associated with the maintenance of an exhaustive list of local programs, simple configuration of the system, and multiple functions to facilitate data reuse. Mobyle also embeds a workflow system engine, enabling the automated merge of successive or even parallel tasks.

- wEMBOSS: it provides a web environment from which the user can access and use EMBOSS tools in a easy-to-use modality. In addition to the user interfac e, wEMBOSS organizes work in projects and the interface allows to use result files from one project as input of others.

- GenePattern provides a web interface for hundreds of tools for sequence variation, copy number, proteomic and gene expression analysis. It uses a client-server architecture to run programs on a single or on separate machines, thus allowing the server to take advantage of more powerful hardware. More importantly, GenePattern ensures the reproducibility of analysis methods by recording the used analytic methods, the order in which methods were applied, and all parameter settings. Moreover, the programming interface allows users to access diverse collections of analyses and visualization methods.

1.2.2 Scientific workflow

For complex bioinformatic analyses a single program is often not enough and the combination of different programs into a workflow is a solution frequently used to produce a new behaviour from a library of available tools. WMS (Workflow Management System) have been developed to allow the coordinate execution of several programs and to give a uniform user interface, while overcoming some problems related to data compatibility. Many of these systems also offer the opportunity to access remotely installed programs, using dedicated web-service solutions. Within this environment, when a client runs a program which includes a remote service, it does not need to be concerned with the inner workings or the programming language used in order to take benefit of its functionality. WMSs often provide links to local and remote programs owned and maintained by multiple groups or scientific organizations. As researchers require more sophisticated tools to solve scientific problems, a scientific workflow system allows to create a sequence of analysis steps, which corresponds to a given computational experiment. For example, a data analysis process consisting of a series of preprocessing, analysis and post-processing steps is a typical scientific workflow. In the bioinformatics field, workflows may involve analysis and transfer of big amounts of data (hundred gigabyte or more) and require the workflow platform to be robust and reliable. The lifecycle of a typical workflow (Fig. 2) is composed of four basic phases:

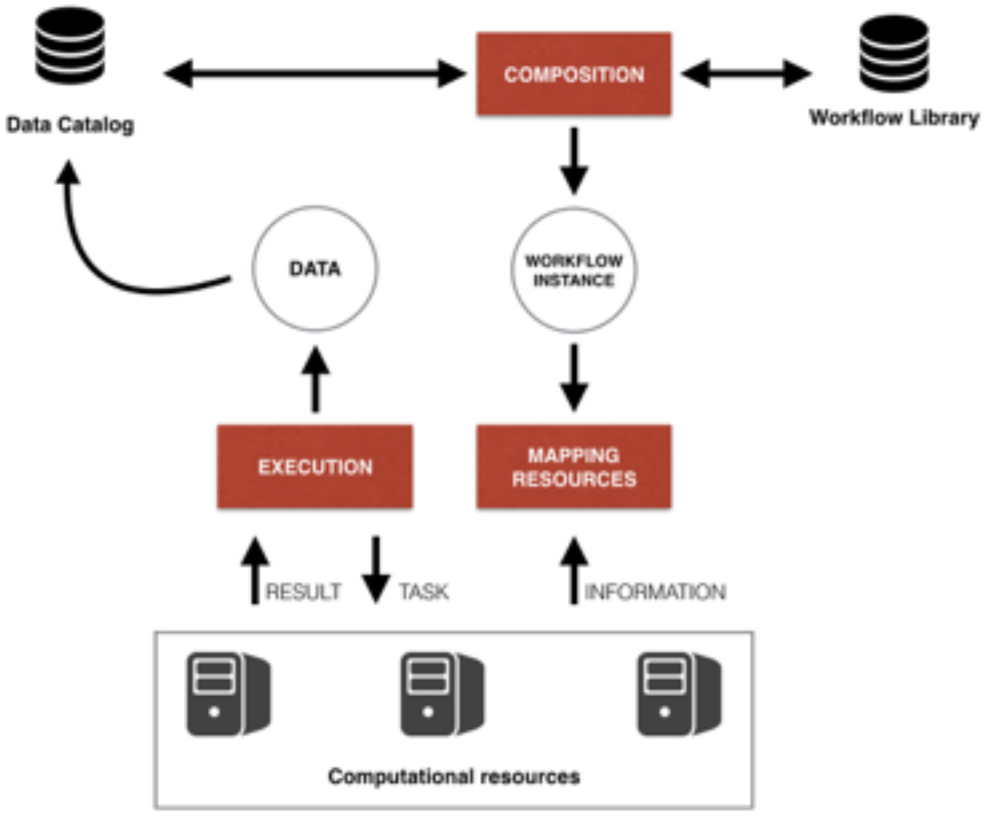
1. the editing phase, which includes the definition of the different processing steps and their sequence, can be done through different means, and is typically based on graphic or text tools. It may take advantage of previously used analyses. This phase allows a user to specify the steps and dependencies also through reuse of templates which support workflow sharing and reuse.

2. Mapping of the processing steps: it implies mapping from the composed abstract workflow to the underlying real resources. Workflow systems can sometimes automatically perform the mapping, but otherwise users can map appropriate resources in a manual matter.

3. Execution is the running of a mapped workflow. This phase depends on the several specific implementations of workflow systems.

4. Metadata storage implicates recording of metadata information during the several step of the workflow lifecycle.

What is remarkable is that during the phase of mapping and execution many optimizations and scheduling decisions can be made. Moreover, the data and linked metadata are recorded in several registries, which can then be accessed to design a new workflow. Although the metadata recording phase has been described as separated phase of the workflow lifecycle, this process is often part of the workflow



execution.

Figure 2. Schema of classic workflow lifecycle.

There are currently many bioinformatic workflow systems; some examples are:

- Kepler, a scientific workflow construction and orchestration system based on the concept of *actors,* which represent operations or data sources involved in a specific workflow. A useful feature of Kepler is the support for hierarchical embedding of a workflow into another one. Kepler's web and grid service actors allow scientists to use different types of computational resources in a single scientific workflow.

- Taverna is an application that allows use and integration of many tools and databases available on the web, especially through services offering programmatic interfaces, such as web services. It allows researchers to build workflows to perform a range of common bioinformatic tasks, such as sequence analysis and genome annotations. Moreover, Taverna allows to define data flows and convert data from one format to another when the services are not compatible with each other. Both local and remote resources can be integrated through this system and the stored information gives a detailed trace of workflow execution.

- Galaxy allows users to manage data and execute biomedical computational analyses, and can be used via public services or via a local installation (in this case it can be customized to meet particular needs). An important feature of this platform is that final users do not need to program or learn the implementation details of the tools they need for specific computational analysis, because web-based interfaces help in performing analyses in a simplified manner. Datasets to be analyzed may be imported by the users into their workspaces from other data warehouses or uploaded through an upload module. The Galaxy analysis workspace is where users perform computational analyses. An essential step in supporting the execution of a workflow is guaranteeing its reproducibility by using metadata storing procedures, to preserve information such as input and output datatypes, used tools, parameter values, etc. When a user executes an analysis, Galaxy automatically generates metadata for each analysis step, to track source of each step and ensure its repeatability. Overall, the use of this web-based workflow system provides advantages such as easy execution of large-scale analyses and display of the results using existing tools such as genome browsers.

1.2.3 Web services

Web services have been available for many years in bioinformatics, with a growing adoption of standard technologies, allowing databases and programs to be accessed as remote resources within workflows and interactive analysis tools. As remotely installed programs, web services have several advantages, such as uniform interface and interoperability, easy maintainability and accessibility, and application level interaction with remotely installed programs. Moreover, web services can be included in pipelines to provide integration of many different tools and applications in a single analysis tool.

Several research groups and institutes provide programmatic access to various data resources and analysis programs via web service technology. The EBI (European Bioinformatics Institute), one of the main providers of this type of services, offering a large collection of programs available through SOAP and REST protocol based web services, which allow interoperability and integration. At EBI, over 150 tools can be used for analysis of data from almost 200 databases, (also available through web services) or for analysis of users data.

A client (typically a program), through these web services, can execute many remote functions or programs classified in application categories, such as those listed in table 1

Category	**Example programs**
Protein Function Analysis	PROSITE, PfamScan, HMMER
Data Retrivial	Uniprot, ArrayExpress, PDBe
Sequence Similarity Search	BLAST, PSI-BLAST, FASTA
Multiple Sequence Alignment	ClustalW, Clustal Omega, Kalign
Phylogeny	ClustalW2 Phylogeny
Pairwise Sequence Alignment	EMBOSS (matcher, needle, stracher, water), GeneWise, Wise2DBA
RNA Anslysis	MapMi, Infernal cmscan
Sequence Format Conversion	EMBOSS seqret, MView, Readseq
Sequence Statistics	SAPS, EMBOSS pepwindow, EMBOSS cpgplot

Sequence Translation	EMBOSS backtranambig, EMBOSS backtranseq
Structural Analysis	DaliLite, MaxSprout
Literature and Ontologies	BioModels, PMC, PICR, QuickGo

Table 1. List of program categories currently available via web-services at EBI.

NCBI (National Center for Biotechnology Information), even though it offers a smaller number of services compared to EBI, is another important provider of remote services. Again, SOAP and REST are the most used protocols, with SOAP more represented. Available services include Entrez search, PubChem, eFetchSequnce, eFetchSnpService, eFetchPubmedService, among others.
A whole and upgraded list of all available bioinformatic web services is available on BioCatalogue (https://www.biocatalogue.org/), which aims to classify all the available biomedicine web services. The number of bioinformatics web services being developed are increasing every day and, as a result, the number of services registered in BioCatalog is currently more than 1100.

1.3 Increasing performance by multiple servers

Choosing the best platform for a given problem requires an understanding of the complexity of the involved data, as well as the memory, network bandwidth and computational constraints. To address complex problems involving great amounts of data, for example, there is a growing interest in multi-processor bioinformatic solutions, each with its strengths and weaknesses. Certain types of data benefit from targeted investments in distributed systems that accumulate memory or disk bandwidth from several servers and clusters, and in some cases, take advantage of expensive supercomputing resources. In some cases, software can benefit from the use of specially designed hardware, such as processors adapted to process large data vectors, as in the case of graphic processing unit (GPU). Simulating protein folding in all-atom detail, for example, is computationally intensive, and programs for molecular modelling are a classic example of a task that can take advantage of GPU hardware.

Both single server image or other parallel platforms play an important role in supporting biological research. From a computational point of view, different architectures can be used, and a number of instruments are used to address the relative lack of computing power for bioinformatics.

- Systems based on multiple servers, where each server provides the same services, allow to get the advantage of distributing requests from multiple users on different servers and intrinsically also provide a degree of redundancy to the system.
- On the other hand, the use of multiple processors allows to take advantage of this additional power to produce powerful applications. In parallel computing more cores and CPUs can be used simultaneously to execute tasks that require huge amounts of hardware resources and can be used to reduce total processing time, by distributing single tasks over many processors. The cooperation of many processors to a single goal is typically made possible thanks to two distinct operations, splitting the computational load into many parts which may be executed in parallel and reconnecting the partial results in order to create the right output. The use of parallel computing has been shown to be a valid way to deal with tougher problems in bioinformatics, as it effectually exploits the available hardware resources. In this approach, software should be modified in order to directly support parallel execution, if more processors are available.
- Multiple computers linked together through a LAN (Local Area Network) can effectively share jobs producing clusters of computational nodes. Many bioinformatic applications can take advantages of a cluster architecture, for example, by distributing the evaluation of a large number of DNA sequences over several nodes to completes a search in significantly less time than by using a

single computer. Compared with the single memory image, clusters are highly scalable and, being based on readily available hardware solutions, allow a substantial reduction in the costs associated with building and maintaining the hardware architecture.

- Grid computing is a distributed computing system where loosely connected resources are coordinated by standard protocols and interfaces to produce parallel processing services. Basically it is based on a combination of networked computers, that may even be separately maintained by independent research institutional, that work together on common computational tasks. Contrary to the cluster architecture, grids are typically more heterogeneous and geographically dispersed and have proven to be an important tool for many scientific fields, including bioinformatics and computational biology. Therefore, many research groups have started to make use of grids and other distributed computing environments, attracted by the idea of joining large-scale computational resources at reduced costs. ABCGrid is an example of a grid computing application in the bioinformatics field, designed to use heterogeneous computing resources and access different bioinformatics applications. The extreme example of distributed computing is found in projects such as folding@home, which aims to distribute load over an extremely large number of small CPUs, taking advantage of a very much parallelised approach.

1.4 Architecture models used in distributed systems

In a distributed system, computers located on a communication network transfer information by passing data packets and instructions to execute programs with specific arguments or to send and receive data. This is a valuable approach to provide services, share data, or store data sets that are too large to fit on a single machine.

Computers in a distributed system can have different roles that depend on the organization of the system. Distributed systems are often based on modular architectures, in which, the components of a system do not depend on how other components implement their behaviour, as long as an interface is specified and the inputs that should be accepted, as well as the outputs that should be returned in response are defined. Main advantages linked to modularity are that:

- the system is easy to change, expand and to understand;
- system errors can be solved by replacing only the malfunctioning components and bugs or breakdowns are easer to isolate.

Different architecture models are used in distributed systems.

1.4.1 Peer-to-peer architecture

A peer-to-peer architecture consists in systems, where load and responsibility are equally shared among all the components (peers) of the system (Fig. 3).

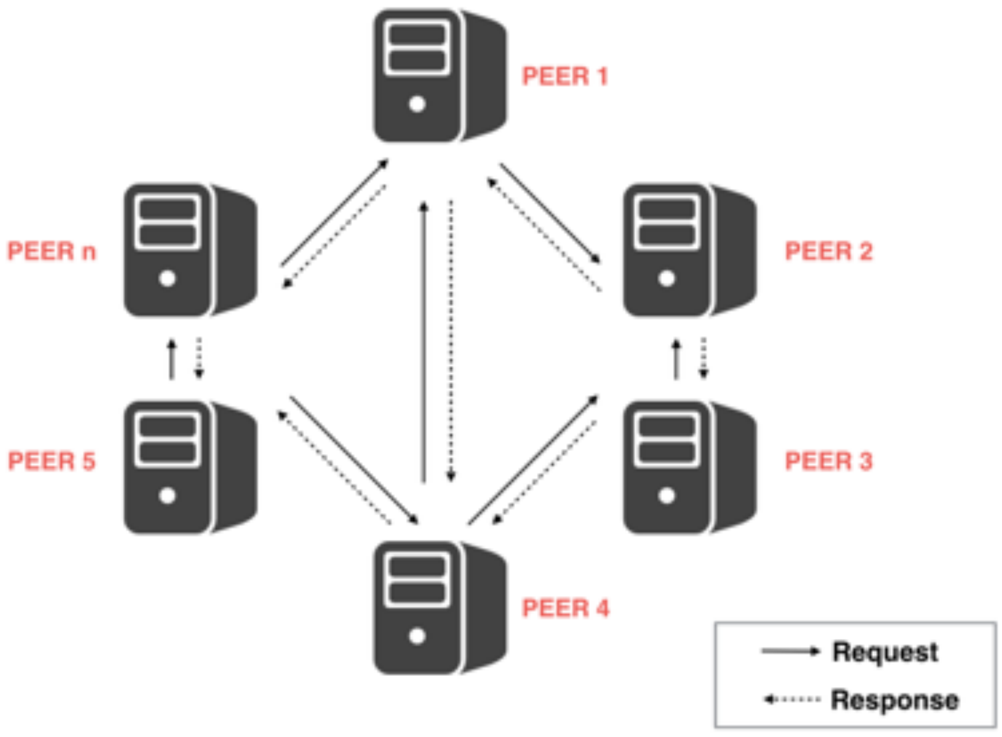


Figure 3. Peer-to-peer architecture.

In this type of architecture, all the involved computers, typically have the capacity to contribute processing power and memory, leading to growing computational resources as the system increases in size. In order to make sure that peers are able to communicate with each other on a reliable basis, peer-to-peer systems may have the support of dedicated components that execute functions, such as maintaining information about the location of different participants.

1.4.2 Client-server architecture

A client-server architecture is based on a single central host (server), that provides a number of services to multiple clients that communicate with it (Fig. 4). In this type of architecture the task of the server is to respond to service requests from clients that use the service. Clients do not need to know how the service is provided, or how the data are processed, and the server does not generally know how data is going to be used. Every machine in this approach can be a server or a client according to the application, maybe at the same time, although often bigger machines are configured as dedicated servers to be accessed by a small or large number of clients. A classical example of this type of architecture is the use of remotely installed programs through one of the methods described in paragraph 1.2, where one central server meets the requests of many clients. The main benefits of the client/server architecture are:

- centralized data: data maintenance is far easier to manage than other type of architectures, because data is stored only on one server;
- easier maintenance: a client cannot access a server during repair, upgrade, or relocation activities, but is otherwise unaffected;
- security: storing data on a server offers better control of security.

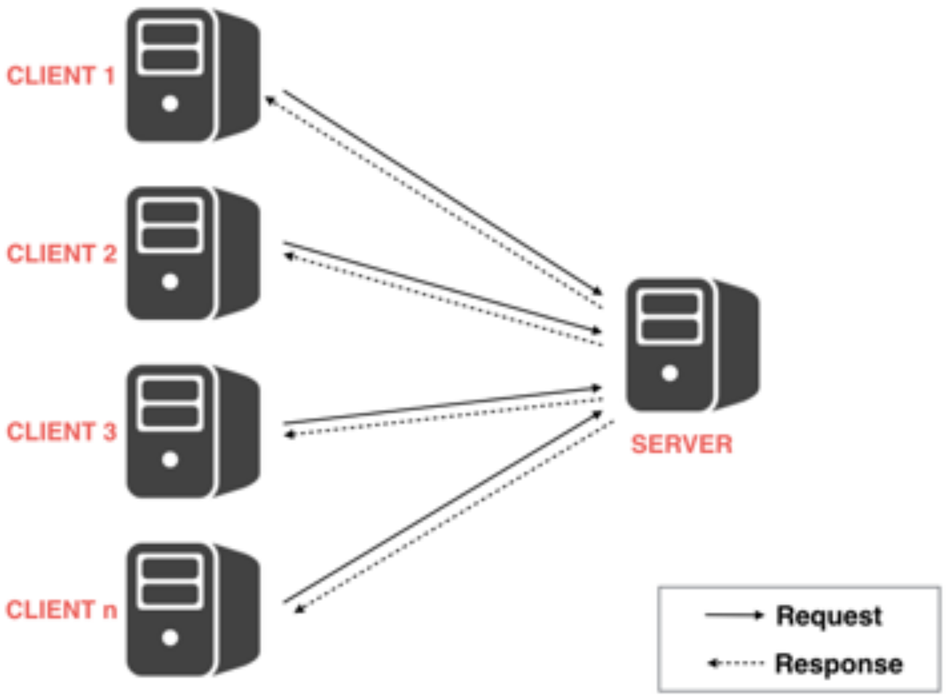


Figure 4. Client-server architecture.

However, the traditional client/server architecture has limitations related to system extensibility and scalability, as its dependence on a central server can negatively impact performance and reliability. In this architecture, resources become limited if too many clients require a given service at the same time and it cannot grow indefinitely with changing demand. Another disadvantage is that the server, is the only component with the capacity to provide the service and is therefore a single point of failure (SPOF) which, if it fails, will break the entire system.

1.4.3 Node communication and message passing

In both architectures, and in general in all distributed systems, nodes communicate with each other by passing data and messages over the network for invoking actions. Communication protocols may act through two principal modality:

- synchronously, when the calling process is blocked until the operation completes;

- asynchronously, when the request only starts the operation and other tasks may be executed by the caller while processing continues. The caller can discover completion by different methods, like, for example, *status* request.

In order to be working, all messages sent over the network must be structured according to a reliable communication protocol, based on a set of fixed rules for encoding and decoding messages. The interacting components must know the protocol in order to communicate with each other. A classic example of message protocols is HTTP (Hypertext Transfer Protocol), used today in many Internet distributed systems.

Asynchronous modes often rely on message passing, where requests are forwarded by a sender to a recipient via the net. Applications based on a set of services that interact with each other by sending messages are built according to a paradigm called Service-Oriented Architecture (SOA), and is often implemented using a service-based technology, such as SOAP or REST. This approach can be used even when the parties requesting and providing services are programs running on different operating systems or written in different programming languages.

1.4.4 Enterprise Service Bus architecture

The need for effective communication between processes running on different machines and often based on incompatible architectures within the same application, led to the development of the concept of Enterprise Service Bus (ESB), a software architecture model able to extend middleware functionality through the connection of heterogeneous components and systems and to offers integration and adaptation of different services. Within this model the point-to-point approach is superseded by a bus with which all different components interact. In this way a lightweight integration solution may be obtained even when many additional systems are added and the number of point-to-point connections required to create a comprehensive integrated architecture begins to increase exponentially. The limits of infrastructures that use a point to point approach have been addressed by EAI (Enterprise Application Integration) solutions, which use various models of middleware to centralize and standardize integration practices. One of the first designed EAI solutions was based on incorporation of all the functionality required for integration (such as all message transformation, routing, and any other inter-application functionality) into central hubs and founded on the principle that all communication between applications must flow through the hub. One of the great benefits of this EAI solution is the ability of integrated applications to communicate asynchronously. This means that an application can do a request and, in the meanwhile the task is completed, continue to work, without waiting for a response. However, since in this model all messages between applicants must pass through a central hub, the latter represents a single point of failure.

In order to overcome these limits a new bus-based EAI solution was proposed, where a central component, the bus, is still used to pass messages between the applications, but the bus architecture resolve the single point of failure problem by distributing some of the integration tasks to other parts of the platform. This new bus-based EAI solution is called Enterprise Service Bus, or ESB. The latter includes several features, such as location transparency, protocol conversion and transformation of messages into a usable format for service consumer, routing, monitoring and administration, security etc. ESB acts as a communication bus in a service-oriented architecture (Fig. 5), translating client requests into the suitable message types and routes them to the various providers. In this scenario, a service requester does not know where the service provider is physically located and an efficient communication between services and applications is produced by a messaging architecture, which takes care of handling protocol conversion, message format transformation, orchestration, routing, etc. The ESB framework also provides methods for integrating web-service based applications and is extensible because new services can easily be integrated into the bus. A fundamental aspect of this architecture is the indirect interaction between service providers and consumers, because the interaction always happens via the ESB, which acts as a message broker between the applications.

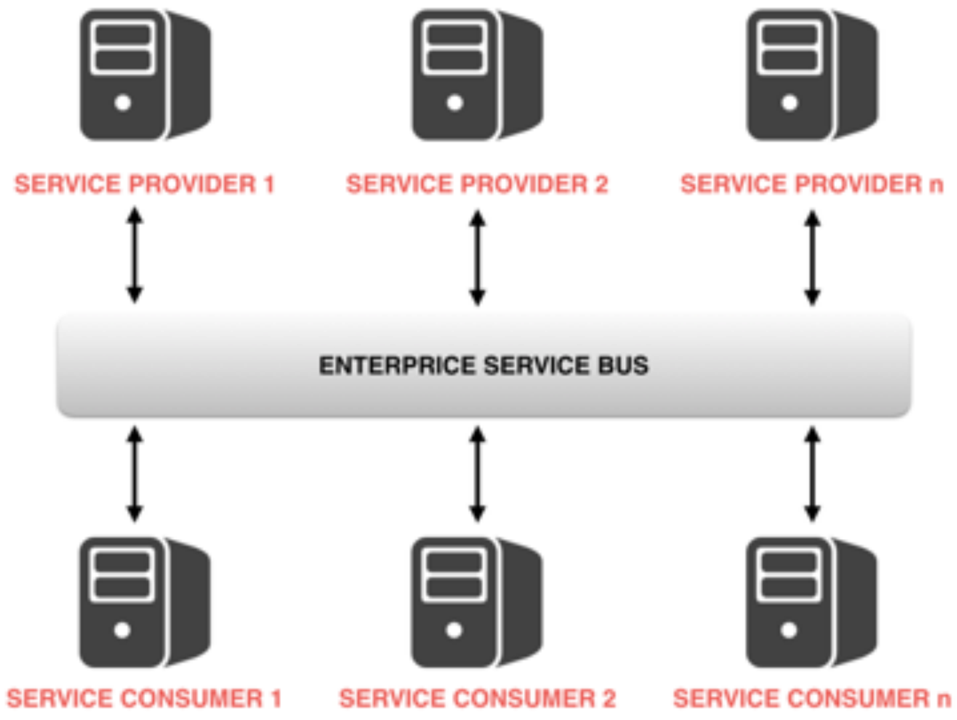


Figure 5. Enterprise Service Bus architecture.

Usually, in a ESB architecture, a common message format is used. Transformations are only executed at inbound entry points to the ESB or, when the flow is at its end, at outbound endpoint. In that way, the internal processing logic of the ESB works with a one format. The user of ESB does not have to work with transport specific formats, given that an ESB implementation is generally able to support multiple different communication protocols, such as HTTP, JMS or FTP. ESB architectures support synchronous and asynchronous communication styles and should be able to poll resources like file systems, databases and scheduler systems.

In an ESB, custom integration services may be created, extended, and reused. Exposed services can be assembled together with specialized integration enablers to form merged services that can be recombined and recycled for various tasks. A typical life cycle for a message that originates outside of the ESB, and goes through the bus to be delivered to another service, uses one or more flows (or sub-flow). Typically, a flow is a flexible mechanism that enables orchestration of services and it can be a simple sequence of steps, or a complex process orchestration with parallel execution, using conditional splits and joins. Flows allow creating solutions that closely match specific requirements. Flow processing can be controlled by message metadata or through the use of an orchestration language. Process flow can also include dedicated integration services that perform intelligent routing of messages based on content. A flow can be represented as a series of message processors (Fig.6), where each message processor is a block of flow. It includes also a message source (the source of messages that are processed by the Message Processor).

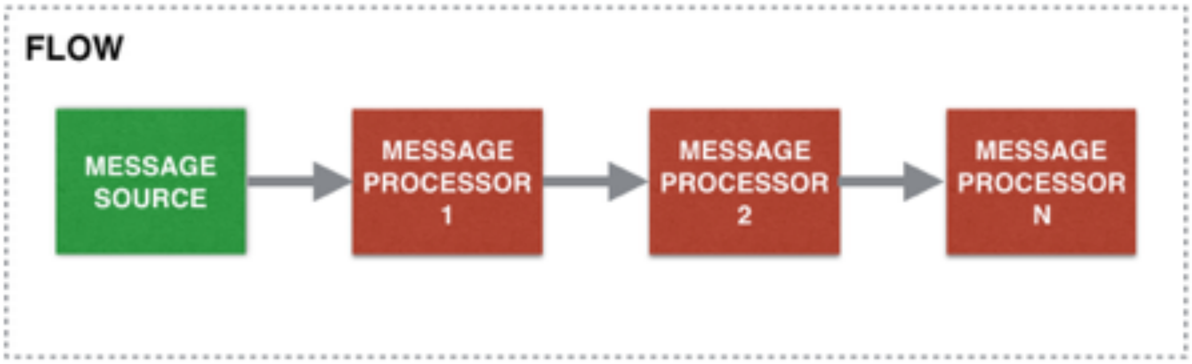


Figure 6. ESB flow schema.

When a message is generated by the message source (or is received from an external source), the flow is started and the message processors are invoked, each one giving rise to another one (in the same order as they are configured). In most cases, an ESB flow can also provide one or more response blocks. If it is used, then any message processors configured in this element are used to process the response message. If it is not used and if none of the message processors performs any processing, then the response uses the result from the last message processor in the flow. Another typical structure of ESB is routers, which allow to route messages to several destinations in an ESB flow. Routers can incorporate logic to analyze and, if required, transform messages before routing takes place. In a flow, for example, a message can be split into some parts and each part can be routed to a different building block. In alternative, more messages can be also combined into a single message before sending it to the next block in the flow. Through routers, messages can be ordered or evaluated to determine which of several possible building blocks represents the next processing step. This flow can then route the message to other flows or sub-flows to perform specific tasks. Flows and sub-flows can process messages either synchronously or asynchronously (Fig. 7).

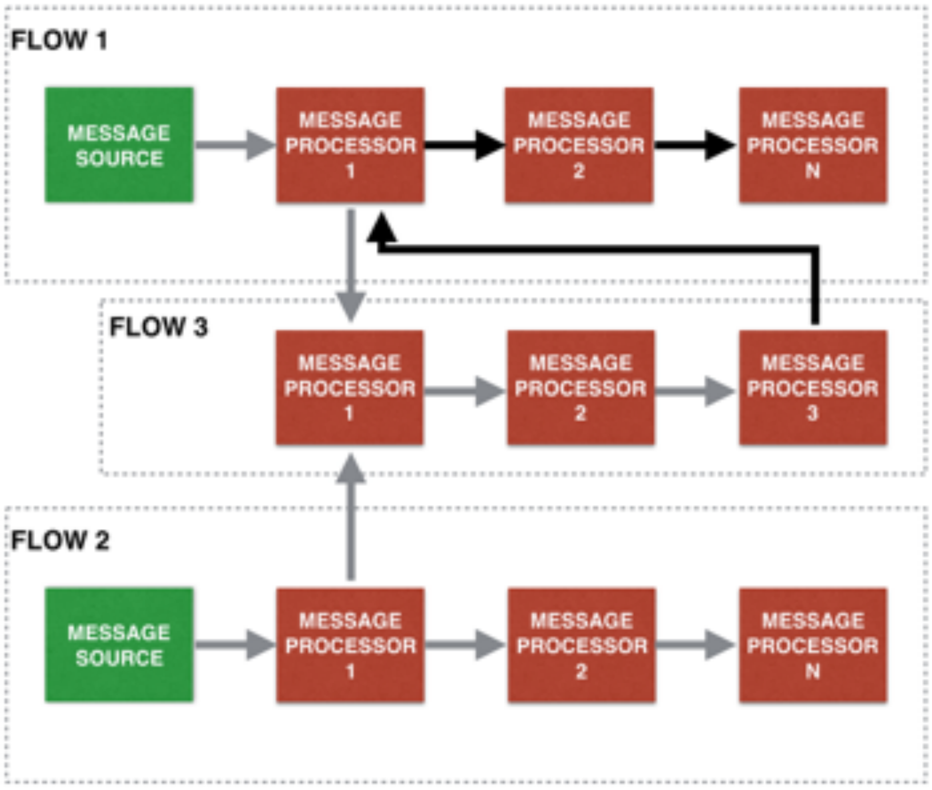


Figure 7. Synchronously and asynchronously flows. Flow 1, trigger of Flow 3, await the response of Flow 3 and then continues the execution (Synchronously flow). Flow 2, another trigger of Flow 3, executes in a parallel manner with the latter (Asynchronously flow).

2. SCOPE

To explore the benefits of sharing distributed computational and storage resources, in this work a platform was developed, able to allow integration of experimental data and bioinformatic services by taking advantage of an Enterprise Service Bus (ESB) architecture. An important requirement of the proposed platform is the choice of a modular organization which allows interfacing with other purpose developed modules and distribution and adaptation of computer services. The modular approach allows easy extension by introduction of new services and scaling by duplication of the most used resources. A redundant approach guaranties smooth maintenance and progressive introduction of changes to essentially all modules.

3. RESULT AND DISCUSSION

3.1 Design of a bioinformatic data processing system based around an ESB architecture

In this work, an ESB platform has been used to design a software infrastructure able to integrate applications of different type and to isolate services and databases from one another by providing a middle service layer with the potential to have develop bioinformatic data processing systems. The Enterprise Service Bus architecture has been used as a platform around which a full data processing system could be centred. This type of architecture is intrinsically able to integrate different service resources through a bus-like infrastructure and provides significant features such as lightweight interfacing and easy expansion. Unlike traditional tools used for data and service integration, an ESB isolates services from one another by building a middle service layer that reduces dependencies by decoupling systems and provides flexibility. The reasons why this type of architecture was used for the development of this system are many and include several important features, such as being:

- lightweight: This is because an ESB is made up of many interoperating services;
- easy to expand: ESB allows easy integration of additional services into their architecture;
- scalable and distributable: ESB functionality can be distributed across a local or even a geographical network. Moreover, because individual modules are used to offer each service, it is much simpler to ensure high availability and scalability by replicating critical parts of the architecture.

3.1.1 An ESB platform for bioinformatic data analysis

The general idea is to let the modules collaborate via a shared bus infrastructure, with which each component is able to interact. In this way, modules are well separated from each other and mostly independent of the language and code of other modules and applications. The central bus architecture results in the integration of the numerous and different applications and services while distributing the requests across different service providers. Using a heterogeneous set of protocols, the platform can be configured to give access to software resources provided by multiple servers. It allows users to run several types of programs with standard interfaces, such as command line execution, through the use of independent modules that can interact with one another through a series of web services. The ESB module is involved in message routing and communication between the components.

In order to select a specific ESB platform able to reach the stated objectives, several features were considered, such as:

- flexible customization;
- usage level in scientific community and businesses;
- support of developer community;
- open source license;
- documentation quality.

Considering these aspects, a short list of several ESB software was initially produced, including:

- JBoss ESB;
- Apache ServiceMix;
- OpenESB;
- Mule.

A detailed evaluation was carried out on the short listed ESB platforms. Tests were based on the evaluation of performance, capacity for integration with external tools and applications and on the ESB introduction impact in terms of communication overhead. In particular, some bioinformatic tools including tools from Emboss, Samtools, BWA and others were used to perform tests on the diverse ESB platform, directed to evaluate response time with and without the ESB. The results of this analysis, together with architectural consideration, led to the choice of Mule ESB, which provides all the above mentioned features and introduced a small delay in response time, which, in our non optimised tests, remained within 2-300 milliseconds.

3.1.2 The ESB based scheduler

The core concept of the ESB is the integration of different applications able to talk to the bus. Thanks to the Bus module and linked components the entire platform allows simplified access to multiple resources even if located on remote servers. Communication with the bus is via a number of alternative protocols and is based on flows built on sub-flows, which together implement the message communication concept. In the present implementation most interactions are based on the HTTP protocol. Requests may be processed in synchronous or asynchronous modality. In the first case, the Bus processes messages just in one thread, which remans active until the execution is finished. In the asynchronous case, a queue is used to unbundle the thread from the outside flow.

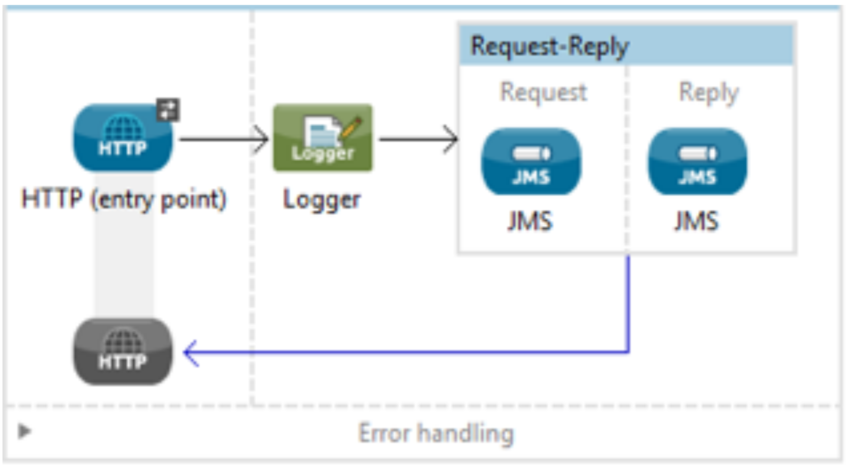


Figure 8. Main internal modules of Bus.

The Bus module is comprised of several internal modules (Fig. 8) and was developed in collaboration with Unlimited Software. The first module provides an entry point to the Bus for external messages and is directly triggered by a HTTP call. In the case of a request for program execution, the following parameters, necessary to execution of the requested programs, are obtained from the call;

- **Package name:** name of requested package (for example, Emboss);
- **Release:** in conjunction with Package name defines the specific package version requested (for example, EMBOSS_6.6.0);
- **Tool:** name of the requested package tool (for example, Matcher);
- **TokenUser:** univocal code that ensures users authentication and used for authorization purpose;
- **Method:** defines the request type;
- **Parameters:** this field contains all parameters that not are included in fields described above (for example the parameters of command line used to execute a tool).

The above described operation is executed (Fig.9) by the "Params Filter" block and the extracted information is passed to the next steps, where the "Token Verifier", interacting with the external Auth module (see below) verifies user credentials and privileges. If user authorization is successful then the "UserCredential" block communicates with the Broker module (see below), verifies if the requested Package is available on the Bus, and selects a suitable resource for execution.

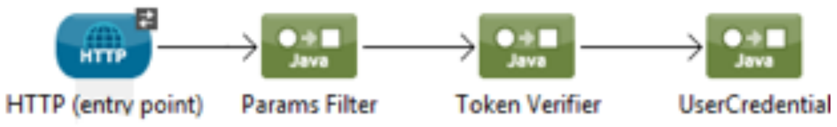


Figure 9. The basic flow of execution within the Bus module.

The value returned from the Broker module is a list of server IDs, able to provide execution of the requested tool. **A serverID** may point to one or more servers and/or cluster nodes; the real server will be selected for execution by another block, which with the help of a scheduler module, identifies a specific cluster and node. Another block, Choice (Fig. 10), is used to choose the protocol used in calling service provider, which can be based on REST or SSH.

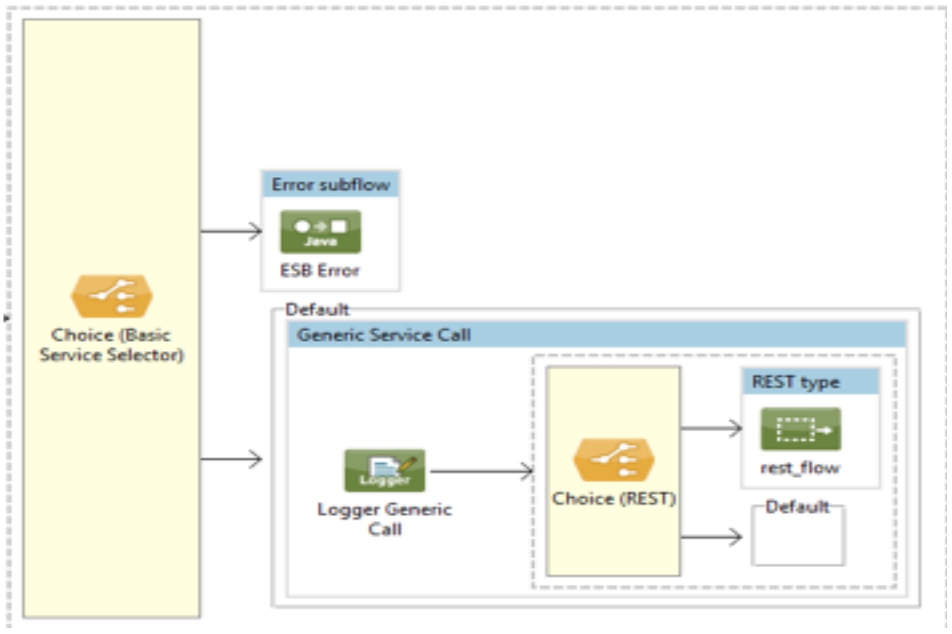


Figure 10. The Choice component of the bus

This depends on two features: original protocol used to execute the first Bus call and available interface (REST or SSH) of the selected servers. If the two are not matching, the Bus uses specific synchronous/asynchronous adaptors/converters. This organization allows to adapt the Bus workflow with the requests of clients. In the REST-based scenario, client requests follow the usual chain of commands:

- Run: to submit requested HTTP call and to get in return a univocal JobID;
- Status: to allow controlling of process status;
- Result: to obtain the result of call.

3.1.3 Client interaction with bus

A client of this system, can be a user that requires a specific service or program to execute a given analysis or an advanced interface such as, for example, a Galaxy project installation, that uses the service providers available to the platform to process data by running typical bioinformatic programs. The client is to be able to do requests to the bus through the described interface to communicate essential information such as program name, version, command and parameters. The request, that consist of a call to the bus, can be done through different modality (synchronous or asynchronous) and using different communication protocol (such as SSH and REST) depending on user needs.

3.1.4 ESBrunner can mimic local execution while uploading processing to the bus

Client interaction with the bus may occur through different modules. As an example, we have developed ESBrunner, a shell script that can run on many LINUX and other UNIX based client computers, where it simulates program execution creating the illusion that the executed program is running on the local machine. ESBrunner gets from the user the provided information (i.e. program name, version, command and parameters) by reading the classic execution command line and constructs and runs the above-mentioned call to the bus. The execution resembles a standard local program:

```
>./emboss_6.6.1/matcher -asequence seq1.fa -bsequence seq2.fa  -outfile output_aln
```

In the above example a standard execution of Matcher from the Emboss package is used to identify local similarities between two input sequences (seq1.fa and seq2.fa). When a user runs a command like this, a symbolic link system associates the name of the package with the ESBrunner script and starts the execution of a series of operations. The first is the extraction, from command line and configuration file, of

the necessary values to build the bus call. After that, ESBrunner executes a call to the bus by using the appropriate synchronous or asynchronous protocol.
ESBrunner is of course only an example of a client. Different clients can be developed with different modality and using different communication protocols supported by the Bus (an example might be SSH). Other clients may interact with the ESB within specific packages, such as web servers or workflow management systems.

3.2 Resource selection and transfer of the request to the service provider

In order to allow the selection of the right service provider that has the ability to run the requested program and meets the additional requirements proposed by the user, the bus relies on additional external modules. In particular, resource selection is carried out with the help of two independent modules, the Broker and the Scheduler, each based on its own SQL database. Resource selection is based on these modules because, when a user requests a program, the bus flow makes a query to the broker and asks whether there is one or more servers with the requested feature. The broker answers with a series of server IDs and then the bus module queries the scheduler for each server ID. In this way the scheduler replies with the address of a free node if available and the identified host will eventually execute the requested command.

3.2.1 A resource broker to select the right service provider

The broker database stores information linked to the various packages and programs installed on servers. When a user requests a program, this database is interrogated and one or more service providers are selected. In this context a provider can be a host or a cluster of servers with same properties. The broker database includes a catalogue of servers on which packages and tools are installed, together with the role permissions associated to each tool. The bus verifies if a requested tool (for example, Matcher) is available for execution by the applicant user by making a query to the broker. In addition to the bus module, that uses the Broker in read-only access modality, administrators, in particular, servers and packages installers, awarded the specific privileges, can catalogue new servers or packages that are installed on the platform through a web-interface.

3.2.2 Node selection based on hardware features and load status

The scheduler database stores the status of each server and node of a cluster. The scheduler, receiving a set of resource requests from the Bus module, selects an appropriate service provider node to execute the request based on several hardware resources evaluated: free ram, free swap, load system average, total running process. This scheduling architecture, operating at the application level, is used on a previously developed scheduler (Boccia et al., 2007) modified to be able to support multiple servers and distribute jobs over a large number of hierarchically organized nodes. The system provides the basic functionality necessary to node selection and service monitoring, and may loosely combine linked computational resources, such as those located in geographically distinct sites. To match the request to the current load and to the resources available on the nodes, the scheduler makes use of the information contained in the related database (a relational database continuously updated with information coming from the nodes). A script, running as a daemon, periodically collects several information, and immediately stores it in the scheduler database. This periodic update is also used as a heartbeat signal, which flags the node itself as active to the system. The information contained in the database is also used to interactively display the status of the clusters. In detail the status of node consists of load level, memory usage, swap memory state, total number of processes, number of running processes, etc. The relational database contains the status information provided by each node, together with info on the recent usage of each node by the scheduling system. For each node, the database stores the information indicated above, which reports the organization of the “nodes” table. Another table, “hosts”, is used to keep track of the node requests, granted to each client host. The table is used to limit the number of node assignments granted to each host and is periodically flushed to guarantee acceptable performance.

3.3 EXEC: a customized server designed for program execution

Program execution within the platform is based on the idea that adaptors may be designed to integrate different type of application servers, even if heterogeneous. However, a specially designed server was developed to use all available features of the platform, including fast access to data and easy configuration. The server software is based on a standard Linux installation, complemented by a software layer able to interface with the bus. This interface was designed in such a way as to permit various access modes to programs, including synchronous and asynchronous execution and commonly used protocols as SSH or REST. The same software layer may also be used to build an interface server able to pass execution requests to an external server which eventually provides the programs (an example can be a front-end/adaptor for EBI - European Bioinformatics Institute - services). In this case the immediate availability of a large number of programs may be granted without installation, while having the ability to configure and maintain the server as in standard installations. In order to meet the described requirements, the interface was designed to:

- support installation of different type of programs;
- provide access to programs through different modalities;
- inform the Scheduler on server status and availability;
- access to installed package information to facilitate some configuration phases.

The software interface toward the Bus interprets the call to the server and builds the final command line for execution of the requested application. The interface is based on a web service constituted from two separated modules, one for REST call-wrapping and the other for SSH call-wrapping. The REST wrapper was developed following the standard of a classic REST architecture, based on the HTTP protocol, where software packages available on the platform are seen as ‘resources’ accessible through a global identifier (URL - Uniform Resource Locator) which looks like this:

http://server/resource/operation/job_id

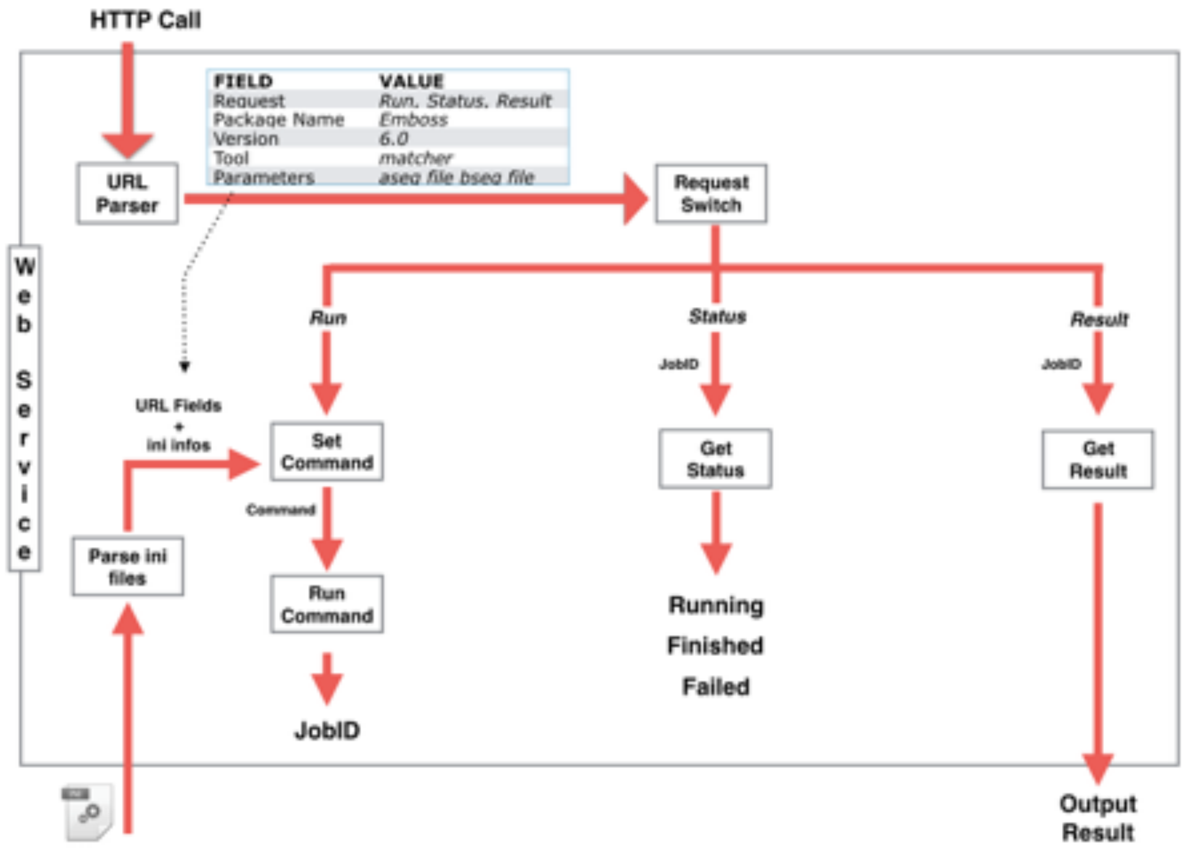
with *job_id* only present if *operation* is a STATUS or RESULT request . The Package Server verifies that the requested application is installed and available on the platform and whether it supports access to files on a shared storage before making the necessary steps to make files such as an input file available on the local machine through a download operation.

Package installation is accompanied by the creation of configuration files, which describe all the features of the package necessary for tool execution, such as installation path and list of authorized roles/users. The following table (Tab. 3) includes the main information stored in the configuration file.

RO	Roles/users authorized to execute the package
EP	Execution path of package
SG	Communication protocol (HTTP, SSH)
RU	Flag that indicatesi if the package support direct read of remote file available through a URL
TO	List of all tool available in the package
SE	Server Identifier

Table 3. Information contained in “exec.ini” configuration file. In the first column, are listed the tags used for each information. In the second column, are listed the descriptions for each tag.

The final execution command line of a particular tool of package is being built on the basis of information stored in the configuration file of a package, in particular of exec.ini files, and on the basis of information wrapped from the HTTP call to this interface, as illustrated in the Figure 13.



FIELD	VALUE
Request	*Run, Status, Result*
Package Name	*Emboss*
Version	*6.0*
Tool	*matcher*
Parameters	*aseq file bseq file*

Figure 13. Schema of Web service for REST calls wrapping

The SSH call-wrapping module is largely based on the same functionality described for the previous module. In particular, they share the routines for local download of remote data and the routines for the management and access to the several package configuration files described above. In this case operating model is based on a BASH script that interprets the SSH call from Bus module and extract the arguments. Afterwards, a second script is executed for integration of functionality linked to configuration files.

This software interface is also involved in the automated retrieval of package-specific informations by the Dashboard module (3.7) to facilitate the configuration phases of package installation when the various packages and tools have to be added to the database of Broker module. To fill the broker database with the catalogue of server packages installed the information on the package may be manually retrieved and annotated, but may also be recovered through an automated procedure that starts by running a command line tool and ends with the creation of one or more structured files containing the retrieved package and tool related information. These files are later used to add the retrieved informations to the Broker database. The procedure checks if a method exists to retrieve information from the specific package and calls it. Each package require a customized method that depends on how the package can be queried to obtain the requested information, for example from a README file or from a manual. Then, if a package-specific method exists, the latter will be used to recover data, if not, the user can write all the necessary information in a proper file.

a hierarchical structure of the configuration files and directories includes common information stored in a general configuration file (exec.ini in Package1, Fig. 12), while specific information is stored in a configuration file for each tool (exec.ini in Tool1 and Tool2 folders, Fig. 12).

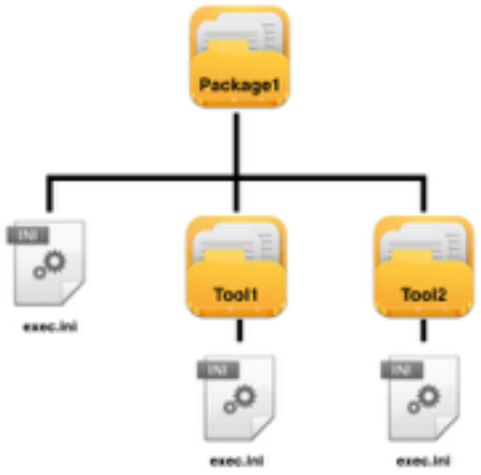


Figure 12. Hierarchy structure of configuration files for a package.

3.4 Access control to system resources

When developing an open platform accessible from different types of users, one of the most important considerations is how users will be authenticated and privileges will be managed. When choosing an authentication protocol, in addition to ensuring access security, it is expected that the last thing a user wants to do is to have to use different username and password in order to access a different functions of platform. An authentication protocol might also be able to provide information about the user, such as a unique identifier, an email address, associated roles, etc. In keeping with these requirements and with the modern access management systems, an authentication/authorization system, based on access tokens and a modified OAuth protocol, was designed and developed with support of an external informatics company. In our system, an access token is a string denoting a specific scope, lifetime, and other access attributes and representing an authorization released to the client. Tokens represent specific scopes and durations of access and are enforced by the authorization server. Access tokens are released to third-party clients by the authorization server with the approval of the resource owner. The client uses the access token to access the protected resources provided by the resource server.

In the context of our platform, when the Bus module takes a call, authentication/ authorization process, that involves an authorisation server, it execute three operations:

- Check for two tokens (user token and program token – for the latter see above) in the client request (this operation is performed by Bus module).
- Verify the validity of two tokens.
- Verify if program token allows user to execute the requested program.

In these operation, the user token is responsible for providing secure access to Bus and to linked services, while program token is responsible for providing certification that program requested is available for a specific user.

The database described above contains information and data linked to all user that can be access the platform an their functionality (such as the possibility of running a program). Thanks to a series of developed web services several module of platform uses this authentication and authorization system to authorize several operation in the system. Example are the login on the Dashboard module, adding of new package or new server etc., operations allowed only if the authorization system, questioned by one of the module, provides the authorization. The communication between several modules and the authentication system is ordinarily executed through exchange of user tokens.

3.5 A centralized/distributed repository for ready data access

Data access and management is an important aspect of any platform able to allow integration of several experimental data and bioinformatic services. However, particularly in a distributed contest, several problems are linked to implementation of a effective solutions. The first of these is related to the availability of data input/ output options on the servers that execute the users requested programs. A possible solution would be to enable client machines (that contains the data input) and server machines (that provide the programs) to access the same data storage through a shared volumes implementation. However, this solution is only applicable inside a local network and is not enforceable to a geographically distributed contest. An alternative solution can be based on a system able to make the necessary files directly available on local filesystem of server that provide the programs. In that respect, it should be considered that a number of programs have the ability to directly access remote files that are available through several network protocols, such as HTTP and FTP. Examples are a few tools of Picard package, a set of programs used to next generation sequencing data manipulation. Obviously, other programs do not include this functionality and then the development of a systems for data access and management are needed which can take into account permission management related to use of data.

In view of these aspects, a system for ready data access was developed in collaboration with an external informatics company (NEATEC), based on a centralized/distributed repository manager (Resource Manager) and on several functions distributed in the other platform modules, in particular in ESBrunner and EXEC.

The Resource Manager allows users to access files, with several levels of authorisation, providing visualization and sharing of remote/local files and the generation of tokens that allow safe access to files through a unique URL identifier. Visualization and sharing of files is available through a web user interface while token generation is done through a suitable web service. Connectors may be created that allow to directly make available data located on a client computer, to visualise the folder files on the web user interface and to create tokens for these user files. After token creation, the corresponding file can be accessed through a specific URL as if it were on the server.

Furthermore, the platform provide an alternative data access modality which is independent from installation of the standalone application on local machine and is based on sharing of volumes between Resource Manger server, user machine and service provider, using the same mount point. Thanks to this last aspect a file path is valid on all involved machines. This system take advantage of several functions of Esbrunner and Exec modules. In particular, when a client execute a command line program with their input files, ESBrunner perform the following operations:

- identify the command line strings that represent data;
- verify if the volume of identified data is a shared volume (thanks to information stored in a configuration file);
- if the volume is shared then the web service for the creation of a token for each file is invoked. This operation use the resource manager system to create

token. Indeed, the latter is able to access the client files because it mounts the same volume of client machine;

- command line strings that represent data are replaced with their tokens (each token is represented by URL);
- call the Bus with final parameters.

If data is not on a shared volume, ESBrunner uses alternative functions to create the tokens, and can transfer the data on the Resource Manger server for token creation.

In all cases, the service providers receive for each input data a URL, containing a specific token for a file, and can operate, using one of several functions of EXEC:

- if the requested program is able to directly access remote files that are available through a URL (for example, Picard) then the URL is directly used as the input;
- if the URL identifies a file available from a shared volume with client machine then the file is directly used;
- in other cases, using the URL, the file will be downloaded on the service provider server before execution.

3.6 The execution flow

The platform has the capacity to allow users to run a program as if it were installed on the local user machine, even without having to change the syntax of command line program. The execution flow (Fig. 13) starts with the invocation, through the execution of a program command line, of the ESBrunner module, that will parse the command line, and do a call to Bus module. This places a call to Auth module to gain associated user roles and to check authorization to execute the requested tool. If the user is authorized then the flow continues with the following steps:

- the Bus module builds and executes a query to Broker module. This query aims to ask if one or more service provider, that meets the requests of user, is available. If the response is positive then the Bus module receives a list of univocal IDs that identify the selected service provider(s);

- upon receiving the ID/IDs, the Bus module, for each ID, calls the Scheduler module following a priority order. The Scheduler evaluates the most appropriate server node for execution, if any, and sends the response;
- upon receiving the selected address, the Bus module builds and executes a call to the service provider, which, builds the final command line to run the requested tool.

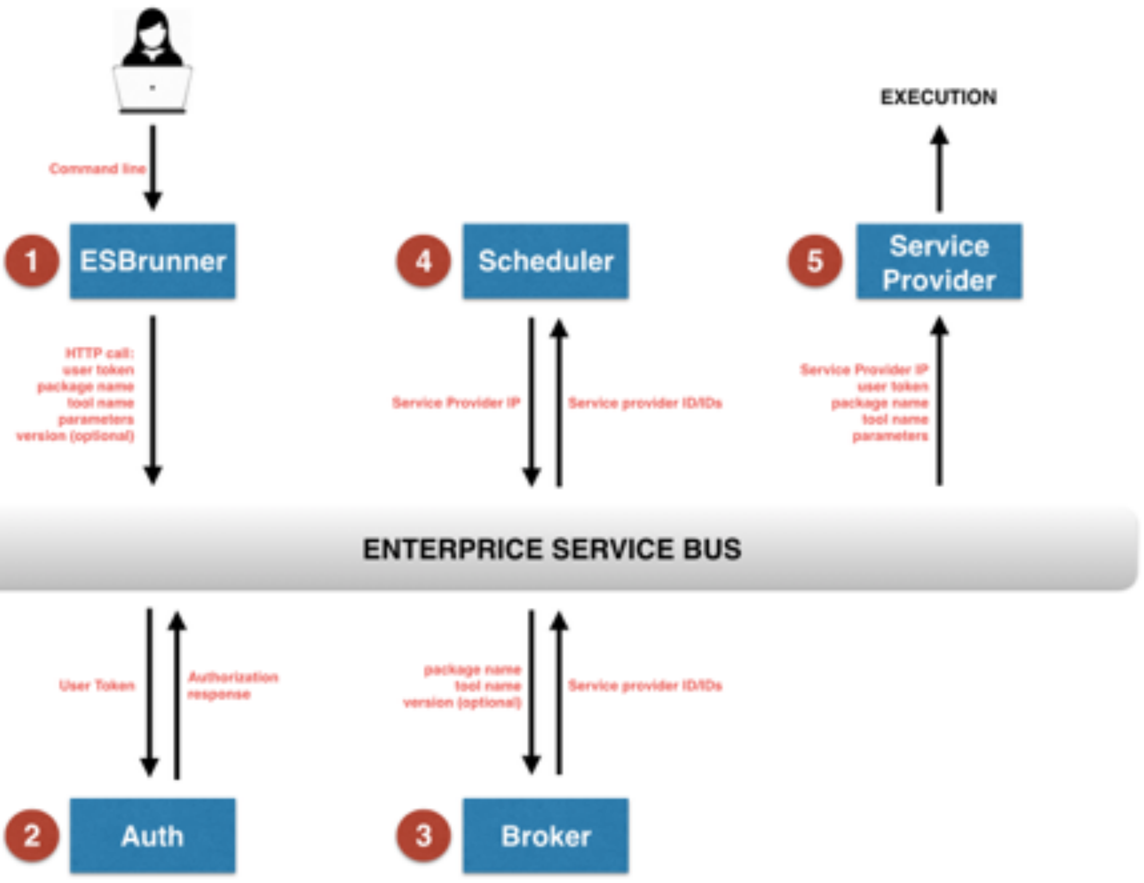
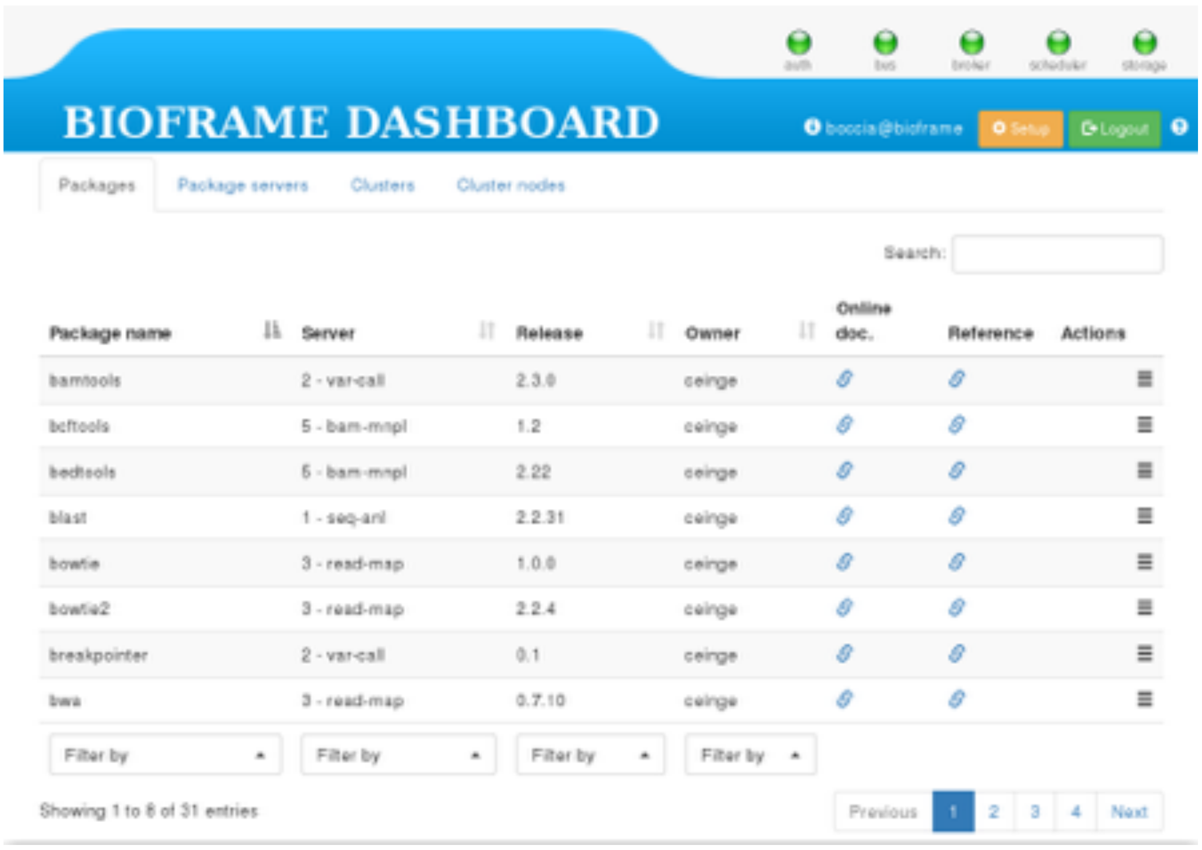


Figure 13. Schema of the execution flow.

3.7 A single configuration point: the Dashboard

The management and configuration of the platform includes several operations that include the installation of new servers and the installation of new packages on the platform. In order to facilitate these operations and to supervise the various components of the system, a web interface, defined Dashboard, was developed. Through this interface it is possible to list all programs installed on each cluster. Most operations may be directly run from the web interface, depending on the user privileges.

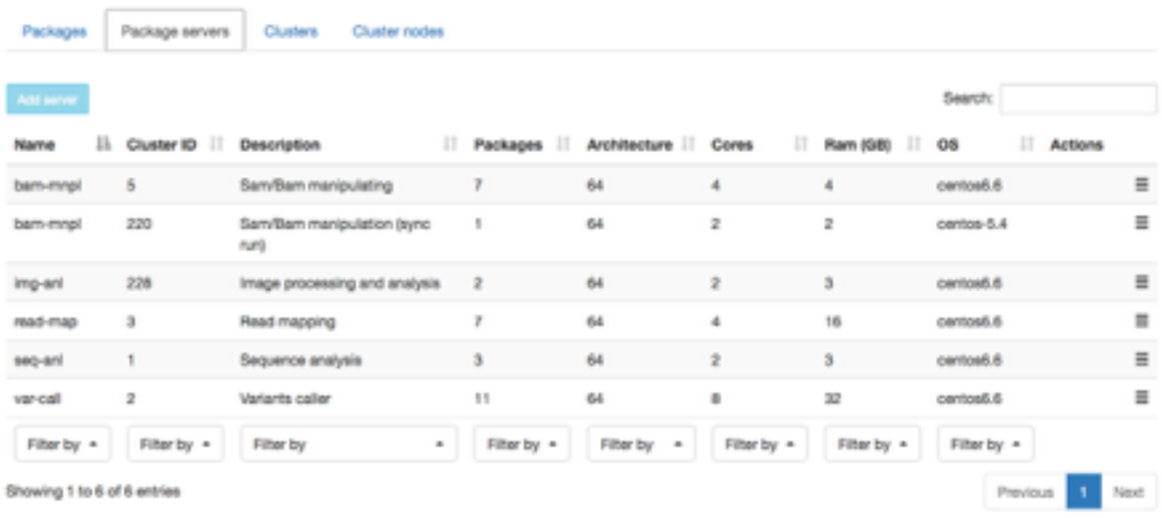


Package name	Server	Release	Owner	Online doc.	Reference	Actions
bamtools	2 - var-call	2.3.0	ceinge	🔗	🔗	≣
bcftools	5 - bam-mnpl	1.2	ceinge	🔗	🔗	≣
bedtools	5 - bam-mnpl	2.22	ceinge	🔗	🔗	≣
blast	1 - seq-anl	2.2.31	ceinge	🔗	🔗	≣
bowtie	3 - read-map	1.0.0	ceinge	🔗	🔗	≣
bowtie2	3 - read-map	2.2.4	ceinge	🔗	🔗	≣
breakpointer	2 - var-call	0.1	ceinge	🔗	🔗	≣
bwa	3 - read-map	0.7.10	ceinge	🔗	🔗	≣

Showing 1 to 8 of 31 entries

Figure 14. The dashboard web interface.

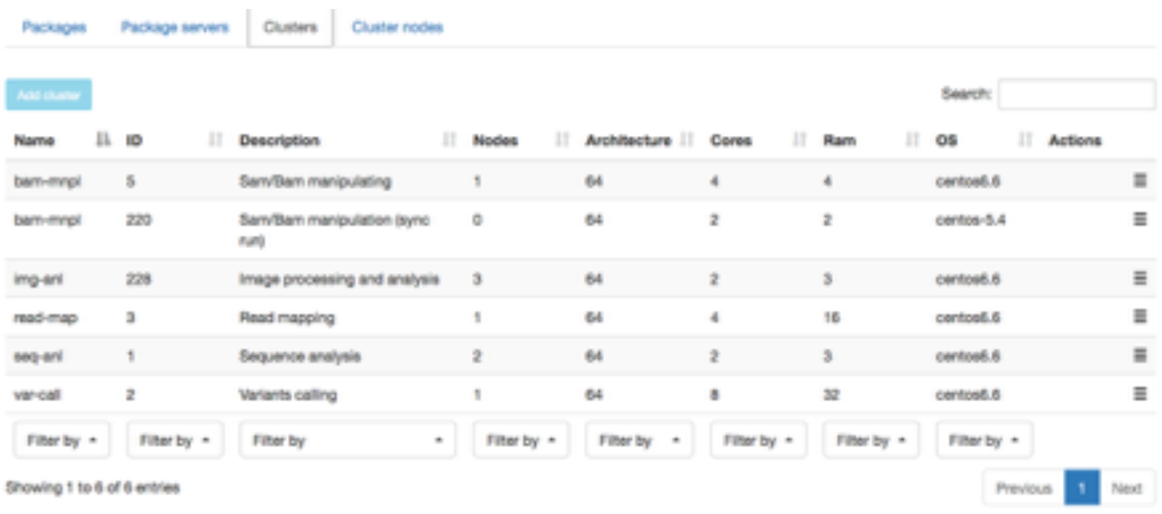
A "package servers" page shows the list of servers available on the Broker database indicating hardware features, such as RAM, cores and architecture, are indicated together with the number of packages installed. As for the programs page, a new server can be configured, if the user has the specific privileges.



Name	Cluster ID	Description	Packages	Architecture	Cores	Ram (GB)	OS	Actions
bam-mnpl	5	Sam/Bam manipulating	7	64	4	4	centos6.6	≡
bam-mnpl	220	Sam/Bam manipulation (sync run)	1	64	2	2	centos-5.4	≡
img-anl	228	Image processing and analysis	2	64	2	3	centos6.6	≡
read-map	3	Read mapping	7	64	4	16	centos6.6	≡
seq-anl	1	Sequence analysis	3	64	2	3	centos6.6	≡
var-call	2	Variants caller	11	64	8	32	centos6.6	≡

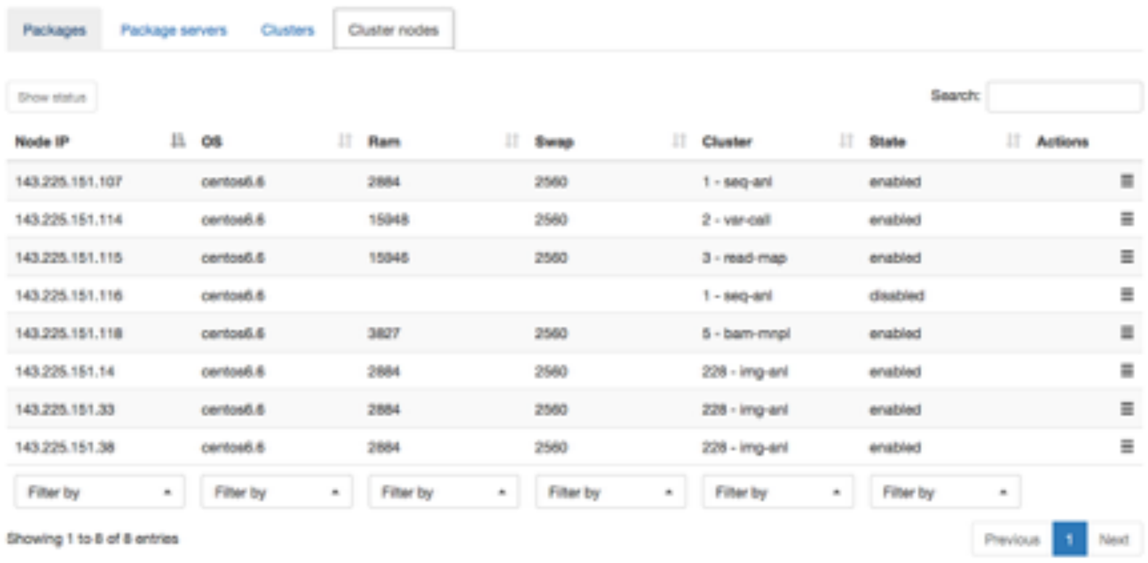
Figure 15. The "package servers" tab of the dashboard web interface.

The “clusters” tab (Fig. 16) contains the list of clusters, which are groups of physical machines, on which the same type of programs are installed. Adding a new cluster, which can include one or more nodes, means creating a new entry in the scheduler database (Fig 17).



Name	ID	Description	Nodes	Architecture	Cores	Ram	OS	Actions
bam-mnpl	5	Sam/Bam manipulating	1	64	4	4	centos6.6	≡
bam-mnpl	220	Sam/Bam manipulation (sync run)	0	64	2	2	centos-5.4	≡
img-anl	228	Image processing and analysis	3	64	2	3	centos6.6	≡
read-map	3	Read mapping	1	64	4	16	centos6.6	≡
seq-anl	1	Sequence analysis	2	64	2	3	centos6.6	≡
var-call	2	Variants calling	1	64	8	32	centos6.6	≡

Figure 16. The "clusters" tab of the dashboard web interface.



Node IP	OS	Ram	Swap	Cluster	State	Actions
143.225.151.107	centos6.6	2884	2560	1 - seq-anl	enabled	≡
143.225.151.114	centos6.6	15948	2560	2 - var-call	enabled	≡
143.225.151.115	centos6.6	15946	2560	3 - read-map	enabled	≡
143.225.151.116	centos6.6			1 - seq-anl	disabled	≡
143.225.151.118	centos6.6	3827	2560	5 - bam-mnpl	enabled	≡
143.225.151.14	centos6.6	2884	2560	228 - img-anl	enabled	≡
143.225.151.33	centos6.6	2884	2560	228 - img-anl	enabled	≡
143.225.151.38	centos6.6	2884	2560	228 - img-anl	enabled	≡

Showing 1 to 8 of 8 entries

Figure 17. The "cluster node" tab of the dashboard web interface.

In the “cluster nodes” tab the single machine are shown here together with the cluster they belong to. Additional characteristics are included, such as the operating system, ram and swap memory (these data are stored in scheduler database). As in the previous cases a new node can be added to a particular cluster.

In order to add a new package to the platform, first the package is installed on a physical cluster node. As soon as the package is usable on the cluster node, it has to be recorded on the Broker by transferring the package-related information to the Broker database with a file, containing package name, release, documentation, reference, and all the information about the server on which it is installed and the regarding the tools and the way to access both the entire package and the distinct tools. In the “Package servers” tab of the dashboard, the user, which has the necessary privileges, can add a new package to the Broker through a automated package installation procedure. These operation are implemented thanks to a series of web services that work as an interface between the Dashboard and broker database.

In order to add a new server or cluster of nodes to the Bioframe platform, several steps are necessary. The first operation involves the installation, on each node of the cluster, of the script that run as daemon and that periodically collects several status informations and stores it in the scheduler database. At this point, through the Dashboard interface it is possible to add first a new cluster from “Clusters” tab (stating the several information on cluster such as name, Ram, Cores, operating system etc. which will be stored on Broker database) and then several nodes of cluster stating IP and owner for each node. These operations allow having a new cluster in scheduler database with a specific ID.

4. METHODS

4.1 Languages

PHP is the main language used during the development of the large majority of modules and web services of platform. PHP is a scripting language. The version currently in use is PHP 5.5.

Bus, Resource manager and Authentication modules rely heavily on JAVA as the main language.

Bash (version 3.2.x and 4.1.x) was used during the development of ESBrunner script.

4.2 Database management systems

The relational model is the pattern selected to implement all database involved in the platform, and PostgreSql is the DBMS selected and installed to manage these database. It relies on a global community of developers and companies and is based on Structured Query Language (SQL), the standard query language for relational database.

5. REFERENCES

5.1 Articles and reviews

- Labarga A. *et al.* (2007). **Web Services at the European Bioinformatics Institute**. (Nucleic Acids Res)
- Fielding, R. T. (2000). **Architectural Styles and the Design of Network-based Software Architectures**.
- Richardson L, Ruby S. (2000). **Restful Web Services**. (*O'Reilly*).
- Samik G. *et al.* (2011). **Software for systems biology: from tools to integrated platforms.** (Nature Reviews Genetics)
- Aisling O'Driscol *et al.* (2013). **'Big data', Hadoop and cloud computing in genomics.** (Methodological Review)
- Casey S. Greene *et al.* (2014). **Big data bioinformatics.** (Journal of Cellular Physiology)
- Ivan Merelli *et al.* (2014). **Managing, Analysing, and Integrating Big Data in Medical Bioinformatics: Open Problems and Future Perspectives.** (BioMed Research International)
- **Yuichi Kodama** *et al.* (2011). **The sequence read archive: explosive growth of sequencing data.** (Nucleic Acids Res)
- Charles E. Cook *et al.* (2016). **The European Bioinformatics Institute in 2016: Data growth and integration.** (Nucleic Acids Res)
- Pieter B. T. *et al.* (2005). **Evolution of web services in bioinformatics.** (Briefings in Bioinformatics)
- Ewa Deelman *et al.* (2009). **Workflows and e-Science: An overview of workflow system features and capabilities.** (Future Generation Computer Systems)
- **Bioinformatics: Concepts, Methodologies, Tools, and Applications**
- Joel T. Dudley *et al.* (2009). **A Quick Guide for Developing Effective Bioinformatics Programming Skills.** (PLoS Comput Biol)
- Elmar Krieger *et al.* (2001). **Folder@Home: distributed computing in bioinformatics using a screensaver based approach.** (Bioinformatics)
- K. Erciyes (2015). **Distributed and Sequential Algorithms for Bioinformatics**. (Springer)
- Ashley Shade et al. (2015). **Computing Workflows for Biologists: A Roadmap.** (PLoS Biol)

- Belinda Giardine et al. (2005). **Galaxy: A platform for interactive large-scale genome analysis.** (Genome Res)
- Robert D. Stevens et al. (2003). **myGrid: personalised bioinformatics on the information grid**
- Boccia A. et al. (2007). **A Fast Job Scheduling System for a Wide Range of Bioinformatic Applications** (IEEE Transactions on NanoBios)
- Ronald C Taylor. et al. (2010). **An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics** (BMC Bioinformatics)
- Anthony R. Et al. (2003). **The discovery net system for high throughput bioinformatics.** (Bioinformatics)
- Alberto Labarga et al. (2007). **Web Services at the European Bioinformatics Institute.** (Nucleic Acids Res)
- Mickael Goujon et al. (2010). **A new bioinformatics analysis tools framework at EMBL–EBI.** (Nucleic Acids Res)
- Jiten Bhagat et al. (2010). **BioCatalogue: a universal catalogue of web services for the life sciences.** (Nucleic Acids Res)
- Hamish McWilliam et al. (2010). **Analysis Tool Web Services from the EMBL-EBI.** (Nucleic Acids Res)
- Weizhong Li et al. (2015). **The EMBL-EBI bioinformatics web and programmatic tools framework.** (Nucleic Acids Res)
- Pieter B. T. Neerincx et al. (2005). **Evolution of web services in bioinformatics** (Briefings in Bioinformatics)
- Duncan Hull et al. (2006). **Taverna: a tool for building and running workflows of services** (Nucleic Acids Res)
- Bertrand Néron et al. (2009). **Mobyle: a new full web bioinformatics framework.** (Bioinformatics)
- He, H. et al. (2003). **What is service-oriented architecture**. (O'Reilly Press)
- Huhns, M. et al. (2005). **Service-oriented computing: Key concepts and principles.** (Internet Computing, IEEE)

- Schmidt, M. T. et al. (2005). **The Enterprise Service Bus: Making service-oriented architecture real.** (IBM Systems Journal)
- Kuehn H. et al. (2008). **Using GenePattern for gene expression analysis**. (Curr Protoc Bioinformatics)
- Sarachu M. et al. (2005). **wEMBOSS: a web interface for EMBOSS.** (Bioinformatics)
- Altintas, I. et al. (2004) **Kepler: an extensible system for design and execution of scientific workflows** (Scientific and Statistical Database Management)
- Bauler, P et al. (2006) **Implementing a Service-Oriented Architecture for Small and Medium Organisations**. (EMISA)
- Schmdt, M. et al. (2005) **The Enterprise Service Bus: Making Service Oriented Real.** (IBM Systems Journal)
- Keen, M. et al. (2005) **Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture**. (IBM RedBooks)
- Ying Sun. et al. (2007) **ABCGrid: Application for Bioinformatics Computing Grid** (Bioinformatics)

5.2 Websites

- "**Web Services Architecture**" (http://www.w3.org/TR/ws-arch/)
- "**EMBL-EBI Web Services**" (http://www.ebi.ac.uk/Tools/webservices/)
- "**NCBI APIs**" (http://www.ncbi.nlm.nih.gov/home/api.shtml)
- **The BioCatalogue: providing a curated catalogue of life science Web services** (https://www.biocatalogue.org/)
- **ncbi_genomes software** (https://github.com/zyndagj/ncbi_genomes)
- **Mule ESB** (https://www.mulesoft.com/)
- **Postgres** (http://www.postgresql.org)
- **OAuth community site** (http://oauth.net/2/)

6. ACKNOWLEDGEMENTS

The work presented here includes parts developed in collaboration with other researchers in the hosting laboratory as well as with two industrial partners: NEATEC and Unlimited Software, and in many cases the described procedures and tools have been designed and developed by working in close association with some colleagues. In particular I wish to mention Angelo Boccia and Antonio Mariniello from the hosting laboratory at DMMBM/CEINGE, Michele di Capua and Antonio Liguori from Unlimited Software and Vincenzo Piccolo and Manolo Forastiere from NEATEC. My contribution to the work has been principally connected to the development of the ESBRunner, EXEC, DashBoard and Scheduler modules and to the planning of most procedures involving the ESB, FileServer, Broker and AUTH Modules. The role played by the industrial collaborators in developing the different modules is specifically mentioned in the text.