INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Cloud-based NGS Data Processing

Alexandre Almeida

João Forja

Projeto Final de Curso

Licenciatura em Engenharia Informática e Computadores

Progress Report

Supervisors: Cátia Vaz
José Simão
Alexandre P. Francisco

May 2016

Table of contents

List of Figures

List of Tables

1 Introduction

Next-Generation Sequencing (NGS) technologies are greatly increasing the amount of genomic computer data, revolutionizing the biosciences field leading to the development of more complex NGS Data Analysis techniques (Shuster, 2008). These techniques, known as pipelines or workflows, consist of running and refining a series of intertwined computational analysis and visualization tasks on large amounts of data. These pipelines involve the use of multiple software tools and data resources in a staged fashion, with the output of one tool being passed as input to the next one.

Due to the complexity of configuring and parametrizing pipelines, the use of NGS Data Analysis techniques is not an easy task for a user without IT knowledge. Moreover, knowing input data can be as much as terabytes and petabytes, pipelines execution require, in general, a great amount of computational resources.

NGSPipes framework is devised to solve the first issue, allowing to easily design and use pipelines, without users need to configure, install and manage tools, servers and complex workflow management systems (Dantas and Fleitas, 2015). In the context of NGSPipes, a pipeline is a series of commands of NGS data-processing tools chained by input and output. NGSPipes offers a DSL to describe pipelines. The DSL's syntax is described following an EBNF notation alike and provides primitives to specify the sequence through which each tool command is executed, to specify arguments, and to chain commands's inputs and outputs. To know each tools's properties and commands, NGSPipes uses tools's metadata which is saved in a user provided repository.

However, as of now, NGSPipes operates only in a single machine, which will most likely lack the necessary resources to execute the pipeline in a reasonable period of time, if at all. Even if users do have access to a powerful machine, there are still problems that need to be solved in order to execute a pipeline as efficiently as possible: NGSPipes does not support parallel execution of pipeline tasks, nor splitting tools's commands input data into multiple fragments to be processed in parallel.

Cloud technologies are sought as a solution to solve the lack of resources of an average computer, a feature that will substantially improve NGSPipes solution, providing users with big clusters of powerful machines to run pipelines more efficiently (Armbrust et al., 2010). NGS4Cloud integrates NGSPipes with the capability of running pipelines in a remote cluster. It also seeks to provide NGSPipes with parallel execution of pipeline tasks, in either different cores of the same machine or different machines, as well as data partitioning. On top of that, we also plan on allowing the multi-core execution of tools's commands that support it. NGS4Cloud tries to achieve these goals while standing to the usability standards of NGSPipes, which aim to keep pipelines easy to use by non-expert IT users.

To implement NGS4Cloud, we are developing an execution engine that analyses the pipeline and builds a graph of tasks that reflects the sequential and parallel nature of the pipeline and is able to deploy those in a cluster of machines governed by the Mesos's batch job scheduling framework Chronos. We will also need to modify the DSL or tools's metadata to allow users to specify data partitioning and multi-core execution of tools's commands. Currently, the input data for the pipeline must be stored locally in the executing machine. This presents a problem in a distributed environment since we don't know a priori in which machine the pipeline task will be executed. We will have to modify this feature to enable using input data from remote locations.

1.1 Outline

This work is is divided in X chapters.

In Chapter 3 we will give an introduction to the technologies that support NGS4Cloud.

In Chapter 4 we will describe each component of the system, how they interact, and which technologies supports them.

In Chapter 5 we provide he data model of our system. We'll address topics like the layout of the task entity, where the data is stored and its structure.

In Chapter 6 we will talk about the plan delineated in the project proposal and how we're following it.

2 Problem Description

DNA sequencing is used in multiple fields like molecular biology, evolutionary biology, medicine and forensics. There has been a big increase on the amount of data produced by DNA sequencing since the appearance of Next-Generation Sequencing methods, creating the necessity to further develop techniques and programs to analyse the data. Due to that, a plurality of tools have appeared to process DNA sequences, or NGS data. These tools's commands are usually executed in structured data workflows and the output of one is the input of others, forming a pipeline. Summing up, in the context of NGS data processing, a pipeline is a series of intertwined computational analysis and visualization tasks on large amounts of data.

NGS data analysis techniques raise some challenges as, depending on the tools's commands and the amount of data processed, the execution of a pipeline may take days or even weeks; the design and parameterization of a pipeline and setup of tools are not easy tasks for users with little to no IT experience, like many biologists; and being able to reproduce a pipeline is rather valuable.

We think a good application should solve the before mentioned challenges by attending to the following objectives:

1. Make the design and parametrization of a pipeline user friendly and easy for non-IT experts;
2. Facilitate the reproducibility of a pipeline;
3. Run the pipelines efficiently, making use of cluster of powerful machines and parallelization techniques.

NGSPipes is a framework to easily design and use pipelines, relying on state of the art cloud technologies to execute them without users need to configure, install and manage tools, servers and complex pipeline systems. NGSPipes fulfils the first two objective, but fails to accomplish the third. NGSPipes does not fully explore the cloud services and remote execution to the maximum, nor the parallelization of a pipeline and its tasks[1], as right now it only allows to execute pipelines on a single machine, running its tasks sequentially, one at a time.

In order to allow a user to abstract from the necessary resources to execute a pipeline and for NGSPipes to be able to validate a given description of a pipeline, it has a component called tool meta-data repository. The tool meta-data repository is a repository which contains information such as: the computational resources needed to execute a tool, the commands a tool allows, the arguments and their types of each command and the docker image (see section 3.1) in which the tool is installed. Since the setup of a tool can be a hard task NGSPipes utilizes docker images to setup and deploy tools.

To facilitate the design and parametrization of a pipeline, NGSPipes offers a DSL that allows users to produce files containing a pipeline description, that can be executed by NGSPipes engine in any machine that runs it. Together with inputs being easily changed, this makes pipelines reproducible. The DSL's syntax is described using an EBNF notation alike and provides the primitives: Pipeline, Tool, Command, Argument and Chain.

The Pipeline primitive indicates whether the tool meta-data repository the pipeline will use is local or remote and its location. The Tool primitive represents what tool will be used. The Command primitive says which command of the tool will be run. The Argument primitive specifies an argument of the command which will be run.

We are creating NGS4Cloud in order to fulfil all the objectives listed above. To fulfil the first two objectives we are integrating NGSPipes and making use of its DSL and tool meta-data repository. To achieve the third objective we will treat remote execution in a cluster of machines and task parallelization as separate concepts.

To explore the remote execution of pipelines in a cluster of machines, we are going to use Mesos (See section 3.2) which allows to efficiently manage clusters of machines, provides fault tolerancy and high availability, and has support for Docker, the technology used by NGSPipes to setup and deploy tools. Since pipeline tasks are batch jobs, we will be using the Chronos framework (see section 3.3), a batch job scheduler framework that runs on top of Mesos and offers a RESTApi for scheduling jobs with dependencies. We will have a software component, the Monitor, responsible for reading a JSON file containing a description of the tasks and its dependencies and using Chronos to schedule jobs corresponding to the tasks.

[1] In the context of a pipeline, a task is the execution of a tool's command.

Regarding the parallelization, we cannot forget that pipelines must remain simple to design and use, therefore we must try to create a high abstraction over the parallelization making this feature as transparent as possible to users, so it does not have a negative impact in usability by non-IT experts. We split parallelization in three levels: Multi-core execution of tool commands that support it; parallel execution of independent tasks of a pipeline; and data partitioning in order to process the fragments in parallel executions of the same tool command.

To support the multi-core execution of a command, we will modify the tools's meta-data to indicate if a tool command supports multi-core execution, and if so, in how many cores; to support the parallelization of independent tool commands we are going to infer from the pipeline description dependencies based on the outputs and inputs of commands; to support the data partitioning we will have to extend the DSL so that users can explicitly split files in smaller files to be processed in parallel and posteriorly joined. Knowing that not all commands can process partitioned data, we will have to add that information to the meta-data of the tools.

There will be a software component, the Analyser (see section 4.3, responsible for interpreting a pipeline description and cross it with the tool meta-data repository, producing tasks (see section 5.1) respective to each command, and a DAG representing the dependencies between tasks. The vertices of the DAG are the tasks, while the edges represent the dependencies. Posteriorly, this DAG will be converted into a JSON object which is saved on a file: the same JSON file that the Monitor reads. This file is called intermediate representation (see section 4.6).

3

Related Technologies

In this chapter it will be given an introduction to the technologies that support NGS4Cloud.

3.1 Docker

Docker[1] is an open-source project which wraps and extends Linux containers technology to create a complete solution for the creation and distribution of containers. The Docker platform provides a vast number of commands to conveniently manipulate containers.

A *container* is an isolated, yet interactive, environment configured with all the dependencies necessary to execute an application. The use of containers brings advantages such as:

- Having little to no overhead compared to running an application natively, due to it interacting directly with the host OS kernel, not existing any layer between the application running and the OS.
- Providing high portability since the application runs in the environment provided by the container; bugs related to runtime environment configurations will almost certainly not occur.
- Running dozens of containers at the same time, thanks to their lightweight nature.
- Executing an application by downloading the container and running it, avoiding going through possible complex installations and set up.

To easily configure the virtual environment that the container hosts, Docker provides Docker images. *Images* are snapshots of all the necessary tools and files to execute an application. Containers can be started from images, the same way virtual machines run snapshots.

[1] https://www.docker.com/, last visited on 26/04/2016

To effortlessly distribute images, Docker provides *registries*. These are public or private stores where users may upload or download images. Docker provides a cloud-based registry service called DockerHub[2].

3.2 Mesos

Apache Mesos[3] is a distributed, high availability, fault tolerant system that allows for multitenancy with the support of containers technologies like Linux containers and Docker. Mesos architecture has three main entities: *masters*, *slaves*, and *frameworks*.

A Mesos cluster can have one or more masters, responsible for managing the resources provided by the slaves. Using Zookeeper[4], Mesos implements a leader election technique to ensure fault tolerance: if the leading master fails, other masters will be ready to replace it.

Slaves are the cluster machines where tasks will be executed. After launching a slave, it uses Zookeeper to register with the current leading master and advertises its available computational resources, such as CPU, memory and disk. A slave can be configured with the list of resources it shall advertise. If omitted it will automatically offer all the available resources.

A framework is a Mesos application responsible for scheduling and executing user provided tasks in the cluster. It can be split into two components: the *scheduler* and *executors*. Using Zookeeper, the scheduler detects the leading master and proceeds to registering with it. The scheduler then starts receiving resource offers from the leading master. If it has pending tasks to be executed, prompted by users, the scheduler chooses a suitable resource offer and launches an executor on the respective slave to run the tasks. Executors are process containers launched in slaves and are responsible for running user submitted tasks. The built-in Mesos executors offer the possibility to launch tasks from shell scripts or docker containers.

3.3 Chronos

Chronos[5] is a Mesos framework specialized in scheduling batch jobs. Similarly to Mesos, it uses Zookeeper to ensure fault tolerance. It is possible to schedule command line jobs and docker jobs through a user interface or through a RESTApi. Jobs can

[2] https://hub.docker.com/, last visited on 26/04/2016
[3] http://mesos.apache.org/, last visited on 26/04/2016
[4] https://zookeeper.apache.org/, last visited on 02/05/2016
[5] https://mesos.github.io/chronos/, last visited on 26/04/2016

be scheduled to run on a specified date and in repeating intervals, with the option to specify the frequency at which the job is meant to rerun. Chronos also allows job dependency scheduling.

4

Architecture

In this chapter we will talk about each component of the system, how they interact, and which technologies supports them.

4.1 Overview

Figure 2.1 is a diagram that shows the system's main components and their interactions. There are three components:

1. Analyser;
2. Tool metadata repository;
3. Monitor.

The interaction begins when a user provides a file with a pipeline's description, specified using the NGS4Cloud DSL, to the system. The analyser inspects the given file and, using the information stored in the tool metadata repository, produces the instructions to execute the pipeline, along with the required computational resources for its execution. These instructions are then written to a file and locally stored, giving origin to an intermediate representation. Having the analyser concluded its job, the monitor is launched. It converts the intermediate representation into jobs's descriptions readable by Chronos and, using Chronos's REST API, schedules them for execution. Having launched the pipeline, the user can now query the system to know its current state. This results in a series of requests from the Monitor to Chronos. When the pipeline finishes executing, the user can download its results by accessing an URI given to him after the pipeline was launched.

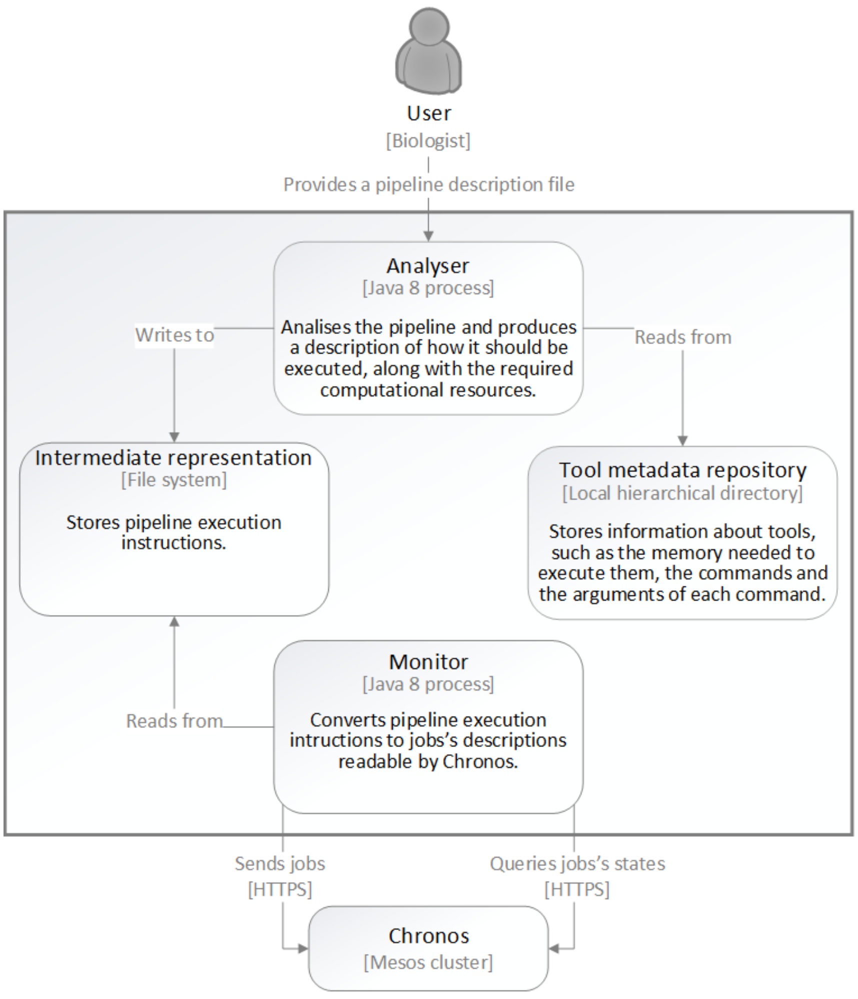
User
[Biologist]
Provides a pipeline description file
Analyser
[Java 8 process]
Analises the pipeline and produces a description of how it should be executed, along with the required computational resources.
Writes to
Reads from
Intermediate representation
[File system]
Stores pipeline execution instructions.
Tool metadata repository
[Local hierarchical directory]
Stores information about tools, such as the memory needed to execute them, the commands and the arguments of each command.
Monitor
[Java 8 process]
Converts pipeline execution intructions to jobs's descriptions readable by Chronos.
Reads from
Sends jobs
[HTTPS]
Queries jobs's states
[HTTPS]
Chronos
[Mesos cluster]

Figure 4.1. Architecture container diagram

4.2 Tool Metadata Repository

In order to allow a user to specify a pipeline without knowing the resources necessary for its execution, the tool metadata repository contains information regarding the necessary computational resources. To validate the description of pipelines, it also contains information about the tools's names, commands and commands's arguments. The repository is implemented as a hierarchical directory system.

4.3 Analyser

It's the analyser who is tasked with mapping the pipeline description into an intermediate representation which has all the information needed for the parallel execution of the pipeline.

The analyser receives a pipeline description and validates it against the information stored in the tool metadata repository. Then it associates to each tool of the pipeline the resources required for its execution with information which is also stored in the tool metadata repository. Through the analysis of the pipeline's description it then proceeds to create a topologically sorted representation based on the dependencies between tools's commands. This representation is stored in a file. The Analyser is developed in Java8 as it reuses components from NGSPipes's engine which is written in Java.

4.4 Monitor

The Monitor parses a pipeline execution instructions file and produces the corresponding jobs description that Chronos understands. The monitor then schedules jobs using the respective descriptions. Besides scheduling jobs, the monitor also queries Chronos to check their progression state. We are implementing the Monitor in Java8.

4.5 Mesos and Chronos

Aside from characteristics like fault tolerance and high availability, Mesos provides support for Docker containers and is currently the target of plenty of development efforts. Having support for Docker is an essential feature for our system since the NGS data processing tools setup is achieved through Docker images: Docker enables tools's automatic setup. By accompanying Mesos development, NGS4Cloud may benefit from future Mesos updates. Hoping that Mesos some day can manage an entire grid system and not a single cluster of machines, our system will also be able to run on a grid system. From a set of Mesos frameworks, Chronos is the only one we found focused on batch job scheduling which offers Docker support, job dependency scheduling and an API.

4.6 Intermediate Representation

Unlike Chronos, there are batch job scheduling frameworks that don't offer an API, but instead offer a library to schedule jobs. These libraries are not necessarily written in Java, the Analyser's development language. In order to decouple the Monitor from the Java programming language, thus allowing for a wider range of scheduling frameworks to be explored, we established an intermediate representation in JSON format that is stored in a text file.

4.7 Conclusion

We believe we've achieved a highly modular architecture where each component is loosely coupled with each other, and therefore easily replaced. This makes our system more testable, extensible and adaptable. Scalability is also a quality due to the use of Mesos.

5

Data Model

In this chapter we will talk about the data model of our system. We will address topics like the layout of the task entity, where the data is stored and its structure.

5.1 Task

NGS4Cloud's main entity is the task. The task entity is the representation of a task in the context of a pipeline. It has information about the tool command to be executed, like the required resources and the docker image to be used, as well as information about the execution itself, like which other tasks it depends upon and the command that will trigger its execution.

5.2 Intermediate Representation

The intermediate representation is a JSON object that contains an array of tasks and an array of input files's locations.

5.3 Data Storage

There are two storages. One for the tool metadata and one for the tools's docker images. The tool metadata is stored in a repository as a hierarchical directory file system. This repository may be local or remote. Using a remote repository allows us to access the data on the repository independently of the machine we are using, and also gives us a backup of the data. The docker images are stored in DockerHub repositories online, being easily distributed by downloading from any machine that needs to run them. Users can add new tools's information to the tool metadata repository and new tool images to DockerHub.

Each tool metadata information will, probably, not occupy over 0.5 MB. Tool docker images are expected to be around 100 to 400 MB. Both the size of the tool metadata file and the docker image depend on the target tool.

Both the tool metadata repository information and the intermediate representation are stored in JSON files and there are schemas to validate them.

6

Plan and progress

Date	Duration(in weeks)	Assignments
03/03/2016	1	- Introduction to Docker and Mesos technologies
10/03/2016	1	- Further study of Mesos technology - Introduction to Marathon and other Mesos frameworks
17/03/2016	1	- Write project proposal - Investigate existent batch job scheduling Mesos frameworks - Study Mesos custom frameworks
24/03/2016	1	- Write project proposal - Further study Mesos custom frameworks
31/03/2016	1	- Deliver project proposal - Study Chronos framework
07/04/2016	2	- Develop monitor
21/04/2016	1	- Test monitor - Wrap up progress report
28/04/2016	1	- Deliver progress report
05/05/2016	3	- Develop analyser
26/05/2016	1	- Test analyser
02/06/2016	1	- Create poster - Prepare Beta version for delivery
09/06/2016	1	- Deliver poster - Deliver Beta version
16/06/2016	2	- Test NGS4Cloud - Finish the report
30/06/2016	1	- Final delivery

Table 6.1. Plan calendar

Since the week starting in March 31st, we haven't properly been following the plan as we took a different approach. We decided to plan the implementation of the system and its requirements before we concretize it. At first it may seem we have fallen behind on schedule since we have not developed or tested the monitor, but that time will be quickly compensated thanks to the planning we've made during the last few weeks. Both the monitor and the analyser are now expected to be developed faster than initially previewed.

References

M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010. ISSN 0001-0782. doi: 10.1145/1721654.1721672. URL http://doi.acm.org/10.1145/1721654.1721672. 3

B. Dantas and C. Fleitas. Infraestrutura de suporte à execução de fluxos de trabalho para bioinformática. Diploma Thesis, Instituto Superior de Engenharia de Lisboa, 2015. 2

D. Greenberg. *Building applications on Mesos.* O'Reilly, Sebastopol, 2016. ISBN 149192652X.

R. Ignazio. *Mesos in Action.* Manning Pubns Co, City, 2016. ISBN 1617292923.

J. Nickoloff. *Docker in action.* Manning Publications, Shelter Island, NY, 2016. ISBN 1633430235.

S. C. Shuster. Next-generation sequencing transforms today's biology. *Nature Methods*, 5(1):16–18, Jan. 2008. 2