Chapter 1

Introduction

1.1 Context: High Performance Computing, Artificial Intelligence and Java

Supercomputing and Artificial Intelligence are among the most important outcomes of the last decades. Both of them have been behind the scenes of many recent discoveries, correctly credited to other classes of instrumentation (e.g. the Hubble telescope), but that required supercomputing and AI as the enabling tools for large datasets processing—usually referred to as "Big Data" [1] [2].

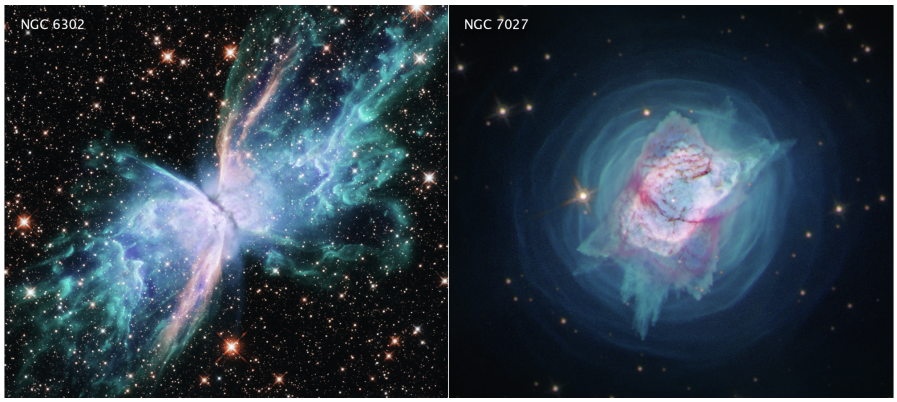

Figure 1.1: Two Planetary Nebulas photographed by the Hubble Telescope

AI applications, from Deep Learning to Data Mining, going through Neural Networks and Clustering algorithms, together with most of the applications in general,

have been switching from a sequential paradigm to parallel and distributed approaches, that best fit the new hardware. The High Performance Computing (HPC) discipline is at the heart of these developments.

HPC is a field of endeavor that relates to all facets of technology, methodology, and application associated with achieving the greatest computing capability possible at any point in time and technology. The action of performing an application on a supercomputer is widely termed "supercomputing" and is synonymous with HPC (T. Sterling *et al* [1, p. 3]).

In this context, the Java programming language plays a marginal role. Languages such as R and Python are much more common when manipulation of Big Data and statistic analysis are the primary goals [3]. However, Java is still in high demand, it is employed in AI and runs effectively on supercomputers. Even if a smaller set of programmers use it for HPC applications, its influence in the AI world is not negligible and it deserves a larger attention to the tools that support its development in such environment.

1.1.1 Java-powered AI and Data Mining

The high and always increasing demand of AI features has affected almost all the programming languages. Research institutions and companies started to invest on AI and Machine Learning [4]. Java, as one of the most common languages, got a bunch of new libraries to enable the developers to access this various world, made of statistics and algorithms. Among all the frameworks for AI, Machine Learning and Data Mining, the ones listed below are probably the most common ones employed with Java. Worthy to be in a resume and capable of figuring in the skills requirement of some tech careers.

Weka The Waikato Environment for Knowledge Analysis (Weka) is an open source software developed at the University of Waikato, in New Zealand. The Weka workbench is a collection of machine learning algorithms and data preprocessing tools, providing a Java library and a graphical User Interface to train and validate data models. It is among the most common Machine Learning frameworks for Java, since it was one of the first ones and it is still maintained [5].

Apache Spark MLlib Apache Spark is an open-source distributed general-purpose cluster-computing framework. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Spark Core provides distributed task dispatching, scheduling, and basic I/O functionalities, exposed through an application programming interface (for Java, Python, Scala, and R) centered on the Resilient Distributed Dataset (RDD) abstraction. RDD is

a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way [6]. Spark MLlib is a distributed machine-learning framework on top of Spark Core that implements many machine learning and statistical algorithms, simplifying large scale machine learning pipelines [7].

Apache Mahout Apache Mahout is a distributed linear algebra framework, written in Java and Scala, whose architecture is built atop a scalable distributed platform. Although Apache Spark is the recommended one, Mahout supports multiple distributed back-ends. The framework features console interface and Java API, that give access to scalable algorithms for clustering, classification, and collaborative filtering [8].

1.1.2 Distributed Java in HPC

The above frameworks are not thought for an HPC environment. The standard implementation of Weka, for example, is designed to run on standard machines (like PCs, laptops or small servers), with most of the algorithms implemented sequentially. This makes it difficult to gain advantage of a strongly parallel architecture like a supercomputer. Spark and Mahout, instead, run both on a distributed platform, which means that they're designed to run on a cluster of different machines instead of a unique system. Java is indeed perfectly suitable to work on a distributed environment, usually using frameworks like MapReduce[1], whose most common implementation is Apache Hadoop, written in Java.

Spark has its own core that work in a similar fashion, Mahout runs on a distributed backend and Weka too can go distributed with some packages, running on frameworks such as Spark or Hadoop [10]. All of them rely directly or indirectly on the map-reduce framework.

Although a supercomputer and a distributed environment, made of nodes connected over a network, are similar from the physical perspective, they're much different when speaking about logic organization. Nevertheless, a distributed system can be deployed on a supercomputer, by keeping the logic organization and taking advantage of its computational power. Such emulated environment can be reached with the use of software containers.

Software containers are a form of OS-level virtualization introduced by Docker. Even if an emulated distributed system can be made of Docker containers, the BSC

[1]MapReduce is a programming model for processing big data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a map procedure, which performs filtering and sorting, and a reduce method, which performs a summary operation [9].

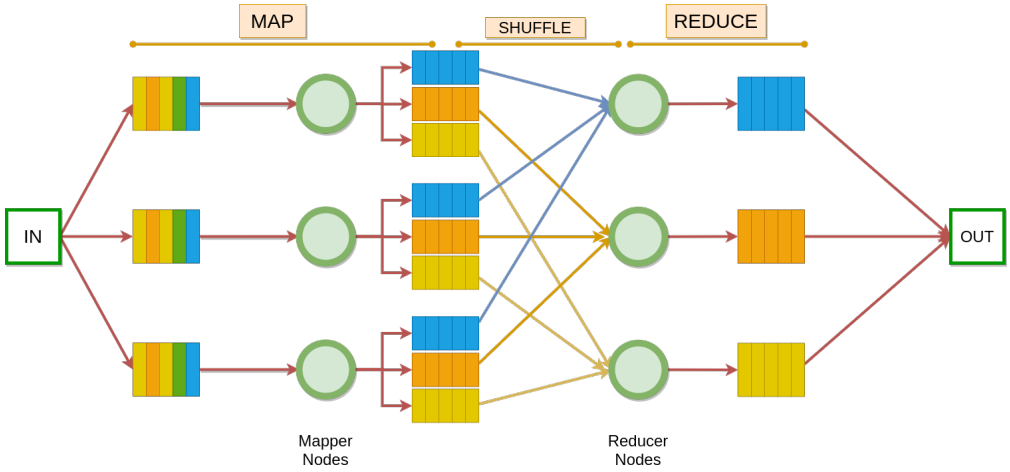

Figure 1.2: How the map-reduce framework works. The nodes can be virtualized using containers.

choice for MareNostrum is Singularity, another type of software containers created as an alternative to Docker for Clusters HPC environments[2].

Singularity enables users to have full control of their environment. Singularity containers can be used to package entire scientific workflows, software and libraries, and even data [11]. Distributed applications based on Hadoop or Spark can run on supercomputers, by using virtualized clusters made of Singularity containers.

1.1.3 Performance analysis

Quoting prof. Jesus Labarta, one of my professors at the UPC, «measurement techniques are "enablers of science". They are present everywhere, and so they are in Computer Science. In the specific case of HPC, they are represented by the performance analysis tools.»

Parallel program performance analysis and tuning is concerned with achieving efficient utilisation of system resources. One common technique is to collect trace data and then analyse it for possible causes of poor performance. The objective is to gather insights, both qualitative and quantitative, in order to increase predictability, build confidence and properly suggest improvements in the software [12].

A great way to describe the observed behaviour is through performance models, which take in consideration multiple model factors based on specific metrics (load

[2]Indeed, that is a common choice in most of the supercomputers worldwide.

balance, communication efficiency, etc.), making it easier to understand where to act.

At the BSC, performance analysis methodologies are mainly based on "traces", generated by instrumenting the libraries and taking several kinds of data from hardware counters, system calls or other low level mechanisms.

1.2 MareNostrum Tools Environment

The work in the present thesis is carried out as an intern at the Barcelona Supercomputing Center (BSC), that has one of the most powerful supercomputers in the world. It is situated right next to the Campus North of the UPC, and the work in the present thesis is carried out as an intern at the BSC.

The BSC supercomputer is called MareNostrum, and it is at its 4th version. With 6,470.8 TFlop/s of max performance, at the time of writing the MareNostrum occupies the 30th position among the most powerful supercomputers in the world, according to the top500 list [13]. It has 165,888 cores, divided on 3,456 nodes, and counts more than 300 TB of memory [14].

Figure 1.3: MareNostrum 4

The MareNostrum software environment is based on the SUSE Linux Enterprise Server Operating System. There are several available modules ready to be loaded, and many of them are developed within the BSC research departments. Among these modules, we find the performance analysis tools developed at the Performance Tools Department of the BSC: Extrae and Paraver.

1.2.1 Extrae

Extrae is a tool that uses different interposition mechanisms to inject probes into the target application, in order to gather information regarding the application behaviour and performance. It uses different interposition mechanisms to inject the probes, and generates trace-files for an analysis carried after the termination of the program [15].

The most common interposition mechanism is the Linker preload "trick". It consists in setting the `LD_PRELOAD` environment variable to the path of the Extrae tracing library, before executing the target program. In this way, if the preloaded library contains the same symbols of other libraries loaded later, it can implement some wrappers for the functions valuable to get data from. This is what Extrae does for most of the instrumented programming models. Another mechanism consists in manually inserting some probes. On top of the instrumented frameworks, Extrae provides an API which gives the user the possibility to manually instrument the application and emit its own events.

Extrae is subjected to different settings, configured through an XML file specified in the `EXTRAE_CONFIG_FILE` environment variable. These settings have the control on almost everything in the instrumentation process. They can enable the instrumentation of some libraries instead of others, the hardware counters, the sampling mode, etc. The tracefiles generated by Extrae are formatted to comply with the Paraver trace format.

1.2.2 Paraver

Paraver is developed to visually inspect the behaviour of an application and then to perform detailed quantitative analysis. It has a clear and modular structure, which gives to the user expressive power and flexibility.

The Paraver trace format has no semantics. Thanks to that, supporting new performance data or new programming models requires only capturing such data in a Paraver trace. Moreover, the metrics are not hardwired on the tool, but programmed. To compute them, the tool offers a large set of time functions, a filter module, and a mechanism to combine two time lines.

Preserving experts knowledge is made very simple. In fact, any view or set of views can be saved as a Paraver configuration file, which reduces the re-computing of the view with new data to a simple loading of a file [16].

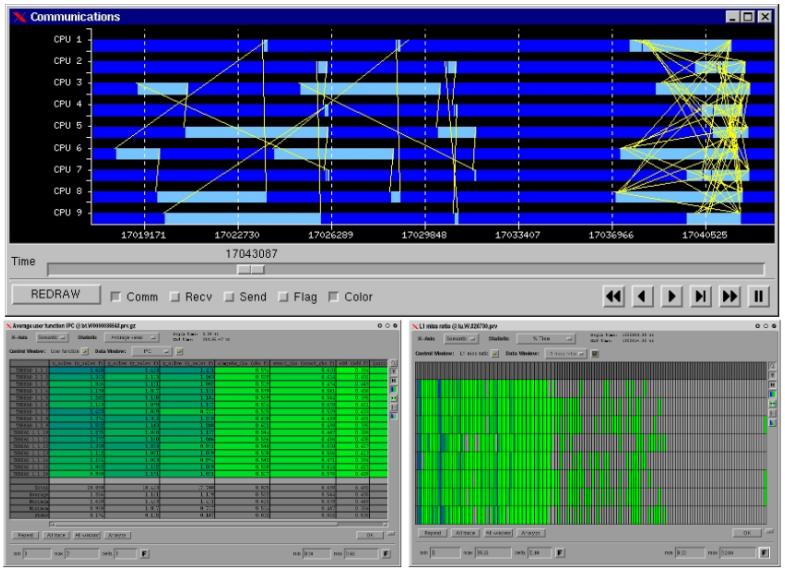

Figure 1.4: Example of traces analysis using Paraver

1.3 Problem Statement and Goal

The final purpose of this thesis was to find some patterns and methodologies that can help analysing the applications based on an HPC Java implementation, by instrumenting the frameworks they rely on. The focus was on the AI frameworks, which represent a large portion of the Java applications that typically run on supercomputers.

Since I developed this thesis as an intern at the Barcelona Supercomputing Center (BSC), the starting basis for the work will be the BSC performance tools, that are not suited for analyzing Java applications. The interest for the company—and so my job as an intern—was to improve such tools (Extrae, mainly) and to gain the right experience in analyzing the typical HPC applications implemented in Java.

1.4 Materials and Methods

If not specified, all the executions reported in this thesis ran on a Dell XPS15 9570, with an Intel i7-8750H CPU at 2.20GHz and 32GB of DDR4 RAM. The programming tools employed in the work have been the following:

- Visual Studio Code for all the `C/C++` code
- Jetbrains IDEA IntelliJ for all the Java/AspectJ code

- Docker to virtualize the OS and the tools environment
- GitHub to store all the code and manage Extrae's pull requests and versioning

All the examples are easily reproducible thanks to the virtualization offered by the Docker image. The image is based on OpenSUSE, provides the BSC tools (Extrae and Paraver) and all the other dependencies (AspectJ, Java, etc.) are installed and ready to be used. The reason behind choosing OpenSUSE is because of the MareNostrum 4 Operating System (SUSE Linux Enterprise Server), of which OpenSUSE—being its free version—is the most similar OS that could be virtualized with a Docker image.

An explanation of the usage of the environment, including the steps to build and setup the image, as well as the instructions on how to run the examples, is present in the Appendix A.