Contents

1 Introduction **3**

2 The COMPSs Programming Model **3**
2.1 COMPSs Components 3
2.2 Runtime Structure 5
2.3 PyCOMPSs Structure 5
2.4 Usability vs Performance 5

3 Collections in COMPSs **5**
3.1 Collection as Input Parameters 5

A Clara.
A mis padres y hermanos. Gracias por haber confiado en mí todos estos años.
A Jero. Aunque ya no estés tu recuerdo sigue conmigo.

1 Introduction

2 The COMPSs Programming Model

COMP Superscalar [1] (and, from now on, COMPSs) is a framework aimed to ease the development of applications for distributed infrastructures. A COMPSs application is typically a normal, sequential application with some special annotations and a few extra function calls in the code.

COMPSs applications can be written in Java, C/C++, and in Python (both 2 and 3). The Python framework is called PyCOMPSs [2]. All the examples and real-world usages in this project will be developed in the PyCOMPSs framework and in the Python language. However, this does not mean that all the features discussed and developed in this project are only available for PyCOMPSs. In fact, given how COMPSs is designed, the implementation of a feature for PyCOMPSs usually implies to implicitly implement it for any programming language.

2.1 COMPSs Components

COMPSs is designed, developed, and deployed in a modular way. This gives some advantages:

- Easier isolation of features
- Partial COMPSs installations are possible (e.g: install COMPSs without PyCOMPSs)
- Components can be individually replaced, leading to faster deployments

An overview of the main COMPSs components can be found in figure 1. These components are also modularized, as seen in figures 2 and 3.
However, this design choice also brings some unwanted issues. The main issue is isolation and concentration of knowledge of some parts in some developers, which leads to unnecessary code replication, lack of coherence of design and implementation choices between different modules, partial feature implementations (e.g: a feature that is only available in PyCOMPSs because it was developed by someone who did not know how to implement it in the runtime), and many other things. All these issues will be adressed and referred to in this document, as they appear and play an important role in our own design choices and implementations.

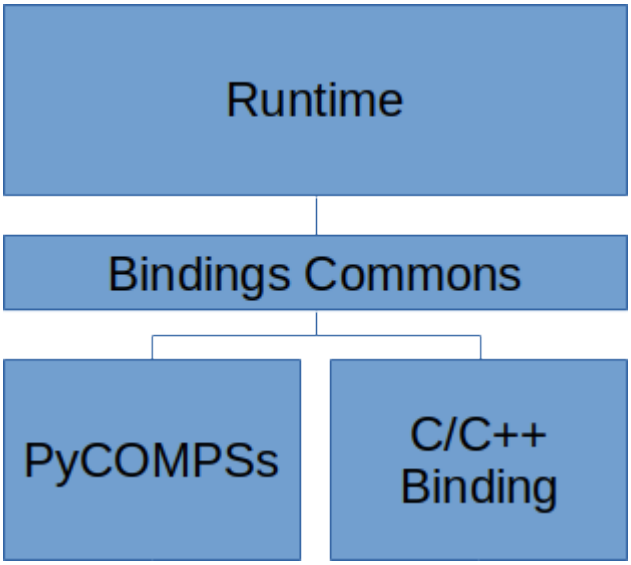

Figure 1: Overview of the main COMPSs components. A line between two modules means that they share messages and communications between them

Figure 2: Overview of the main Runtime components. A line between two modules means that they share messages and communications between them

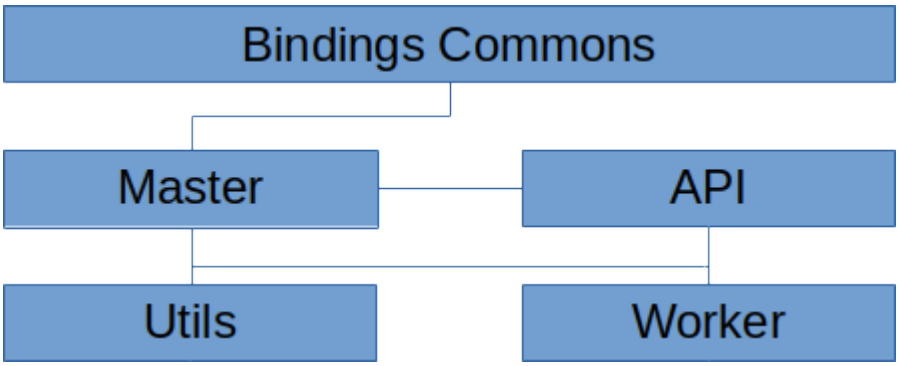

Figure 3: Overview of the main PyCOMPSs components. A line between two modules means that they share messages and communications between them

2.2 Runtime Structure

2.3 PyCOMPSs Structure

2.4 Usability vs Performance

COMPSs has two goals: to give the not-so-expert user an easy way to make their sequential applications run in distributed environments, and to do it as efficiently as possible. Many improvements in the COMPSs framework are aimed to improve only one of these two aspects. For example, any improvement in the communication library may improve the performance of the user application, but the user will still face the same limitations and problems when using COMPSs. Adding an automatic return completion, in case the user forgot to annotate the return value of some task, may save the user a lot of debugging time, but it will have no impact in the performance of the user application.

COMPSs is developed and mantained by a research team in a research center, so it may be natural to think that most of the efforts and improvements are aimed to test and develop methods, models, and algorithms that improve performance, memory usage, minimize network transfers and so on. However, COMPSs is also used by other research teams as a *tool* for their own purposes. Some of these teams intend to run exotic, old, complicated applications in distributed environments. Also, these teams are usually composed of researchers from fields differents than computer science, so a lack of knowledge in parallel and distributed applications should be expected. The user-oriented features intend to help these research teams, making their life easier in the very complicated world of distributed computing.

This project tries to bring something that improves COMPSs in these two directions: give something to the user that makes his life easier while making COMPSs more efficient. We do not intend to limit ourselves to give the user a way to pack some parameters in a collection. We see this feature as an opportunity to give COMPSs additional intelligence that may help to improve the performance of the framework.

3 Collections in COMPSs

3.1 Collection as Input Parameters

References

[1] R. M. Badia, J. Conejero, C. Diaz, J. Ejarque, D. Lezzi, F. Lordan, C. Ramon-Cortes Vilarrodona, and R. Sirvent, "Comp superscalar, an interoperable programming framework," *SoftwareX*, vol. 3, 11 2015.

[2] E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R. M. Badia, J. Torres, T. Cortes, and J. Labarta, "Pycompss: Parallel computational workflows in python," *The International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 66–82, 2017.