Pingo: a Framework for the Management of Storage of Intermediate Outputs of Computational Workflows

by

Jadiel de Armas

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved October 2016 by the
Graduate Supervisory Committee:

Rida Bazzi, Chair
Dijiang Huang

ARIZONA STATE UNIVERSITY

December 2016

ABSTRACT

This is an example abstract that I will have to modify later on. I will write my abstract at the end, when I am done with my research.

DEDICATION

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

LIST OF TABLES

LIST OF TABLES

Chapter 1

INTRODUCTION

The scientific process increasingly benefits from the use of computation to achieve advances faster. Many times these computations can be naturally broken into steps, where each step may filter, transform or compute on the data it receives as input from another step. Workflows have emerged as a paradigm for representing these computations. Many seminal works on the topic of workflow managing systems began to appear in the mid 2000's**???**, and many workflow systems were developed, such as the e-Science project**?**, Kepler**?** and Taverna**?**.

The scale of computations have been growing with time, and the ability of the systems cited above to process large amounts of data and to execute the placement of task execution on a distributed environment is still very limited. Pegasus**?** is a more recent workflow management system for scientific applications. It enables workflows to be executed both locally and on a cluster of computers in a simultaneous manner. It has a rich set of APIs that allow the construction and representation of workflows as Directed Acyclic Graphs (DAGs). It also has more advanced job scheduling and monitoring facilities than previous systems. But still, it cannot meet the scalability demands of today's scientific computations.

Orthogonal to the development of workflow management systems, many distributed systems have been designed and developed to meet the growing demands of computation. One of such systems is Apache Hadoop, which is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Apache Oozie**?**

is Apache Hadoop's workflow and scheduling system. Its workflow definition API rivals that of Pegasus, while it takes advantage of the superior scalability of Hadoop.

Chapter 2

FOUNDATIONAL DESIGN PRINCIPLES OF THE SYSTEM

2.1 Keeping provenance of the data

2.1.1 Actions

The definition of what an action is will dictate the possibility of designing practical functionality that will allow us to optimize the time spent computing a workflow. In essence, that practical functionality refers to skipping certain computations of actions because we have previously computed the datasets that are the output of those computations.

Simplifying things, we can think of an action as a pure function $f : A \to B$. Suppose that we are asked to compute $f(a)$. If we have previously computed $b = g(c)$, and we can determine that $g = f$ and that $c = a$, then we can skip the computation of $f(a)$ (if it is that we still have b among us. Determining if a and c are equal to each other is trivial, but it can be time consuming if the inputs are big. Determining if f and g are equivalent is more difficult and has deep theoretical ramifications (it is computationally imposible in the most broad sense, since the Halting Problem can be reduced to this problem).

Also, comparing our actions to pure mathematical functions is not completely accurate. **?** says that a pure function is a function that "always evaluates the same result value given the same argument values. The function result cannot depend on any hidden information or state that may change while program execution proceeds." In some cases, as we will see in the next chapter, there will be nothing stopping our actions from reading values from environmental variables.

```
{
   input: ["/path/to/input1", "/path/to/input2"],
   executable: ["/path/to/executable"],
   inputParameters: [ {key: "arg1", value: "Hello"},
                                    {key: "arg2", value: "World!"}]
}
```

Figure 2.1: Hypothetical example of description of an action

Because of these difficulties, we need to devise a more practical approach to determine the equality of two actions. That approach needs to be derived from a closer look to the functionality that we want our ideal system to embody. The system will receive descriptions of actions as structured text. As Figure 2.1 shows, those descriptions are simply a collection of parameters such as: path to input folders, path to the executable of the action, extra input parameters, etc.

2.1.2 Workflows

2.2 Bounded storage space

2.2.1 The History of Workflows

2.3 A multi-user multi-component system

APPENDIX A

RAW DATA

BIOGRAPHICAL SKETCH