I
Introduction

Cloud computing has gained a large amount of interest in the IT industry as well as in academia. The US National Institute of Standards and Technology (NIST) provided a definition which captures agreed aspects of cloud computing [84] such as its characteristics, deployment and service models. Dealing with huge amounts of data is an important problem when building large scale systems. As a result of such necessity, different scalable data management solutions have been developed. Major enabling features of such systems are scalability, elasticity, fault-tolerance, self-manageability, and the ability to run on commodity hardware.

An important contribution to large scale data processing has been the programming model MapReduce [24]. It was designed to be fault-tolerant, to have high-availability, and to be able to run on heterogeneous hardware. That is the reason why it has become almost a *de facto* standard for massive dataset computations in the cloud. However, the amount of code generated to perform simple data management tasks (e.g. projections, filters, etc.) can become a problem. For instance, programming joins in Hadoop is still a problem for users even though joins are one of the most critical operations in data processing. Along with studied problems such as estimating result size, join selectivity, new challenges come along the way when using MapReduce e.g. data localization, data skewness, and others.

Projects such as Pig [90] and Hive [102] try to alleviate this problem by creating an abstraction layer on top of MapReduce. Hive is a data warehouse which uses Hadoop Distributed File System for storing its data, and a language similar to SQL to access the data, HiveQL. Pig works on top of an open-source version of MapReduce, the Hadoop framework [30], using a high-level programming language called `Pig-Latin`. Pig-Latin resembles some similarities with `SQL` because they can perform some similar operations from relational algebra, but they aim for different things. Pig-Latin allows expressing transformations as a sequence of steps, and on the other hand `SQL` describes the desired outcome. This characteristic is considered an advantage because writing a sequence of steps comes natural to developers and allows them to

write more complex data flows instead of thinking on plain MapReduce, or trying to express their data-flows with a data access language.

These high-level programming languages help improving the usability of `MapReduce`, but they still need further improvements. For example, cost-optimizers are still work in progress, most of the data is stored without maintaining metadata about it, among other features that exist in traditional databases. We would like to explore different approaches to extract the relationships between input parameters and measured performance i.e. query execution time. We hypothesize that using a statistical model we could extrapolate *"what-if"* workload scenarios by analyzing an input workload, and measuring its performance in order to predict it.

However, there are several constraints and challenges we must get around to successfully construct a model able to characterize the performance of a system. This dissertation aims to start such work in order to improve the usability of systems such as Pig, and Hive by using a statistical method for modeling join operations' execution time and then being able to predict them. The construction of such a model is done by setting up a training data set for it, and then performing cross-validation to verify it.

One of the challenges encountered in this research work was the lack of a data set to evaluate specific operations in large scale systems. We used a database benchmark, `TPC-DS`, to simulate real data with real data distributions. Nevertheless, queries proposed in `TPC-DS` are not suitable for systems based on the `MapReduce` programming model. To overcome this, we translated the relations described in the `TPC-DS` into `Pig-Latin` queries to specifically evaluate join operations. This, among others problems will be discussed along this dissertation.

This Master's thesis is structured as follows: Chapter II presents an outline of background information in order to facilitate the reading of the rest of the thesis. Concepts such as cloud computing, MapReduce, and others are explained in this section. In the following chapter III, the big picture of a whole research project is discussed and the objectives for this specific work are explained. In Chapter IV, the approach and design decisions are explained. In addition to this the results obtained are discussed as well as the insights gathered from them. Finally, Chapter V presents the conclusions of this work, and also proposes some future work in order to keep exploring different approaches to the problems stated in this thesis.