III Problem Statement

In this chapter, we put in context the problem this dissertation tries to solve by briefly describing how the *big data* problem appeared, and how it is being handled by different companies, and also by the academia. We also describe the cloud data management solutions to the *big data* problem, and how these solutions are related to the `MapReduce` programming model. Different research initiatives to improve `MapReduce` usability are described in this chapter. Therefore, we are able to place our work among these initiatives. Our initiative consists in creating a model to be used with the Pig system to help the user while using it. Finally, we explicate the main ideas that motivated this project.

III.1 Background

When dealing with huge amounts of data, many of the typical solutions stop being suitable for all cases. For example, many of the web companies such as `Google` [63], `Yahoo!` [66], `Facebook` [62] and `Amazon` [61], had to change the way they managed their data because traditional approaches were becoming unmanageable at web scale.

A cloud-like environment could be used and take advantage in this way of large computer clusters offered by third-party companies. Abadi [1], and Agrawal et al. [4] discuss some of the opportunities and limitations of data management in such environments. In particular, Agrawal et al. point out that we should evaluate carefully the design choices taken while building cloud systems before deciding one in particular. The authors also affirm that we should notice the design decisions avoided, and evaluate the reasons why they were not considered.

Abadi [1] discusses what a cloud database should fulfill in order to be considered as such, about the types of applications that are more suitable to be deployed in the cloud, and most importantly he discusses about cloud environments characteristics e.g. data being stored at untrusted hosts, data being replicated, maybe across large geographic distances, and that compute power is elastic only if the workload is parallelizable. This means that cloud environments are more suitable for computing paradigms such as MapReduce because those paradigms' main idea is to send the computation to the data, and not the other way around. This is an important advantage because moving resources through the network is still an expensive operation. His work also states that MapReduce programming model satisfies some of the desired properties such as fault tolerance, capacity of running in heterogeneous environment and efficiency.

Due to the fact that MapReduce has become almost a *de facto* standard for massive dataset computations in cloud-like environments, there is special interest of the database community in integrating it with traditional research done in the field. That means getting optimizations already researched in the database field into this new type of big data solutions. Research works like HIVE [34], Pig [31], HBase [101], Cassandra [76], among others initiatives, have been started towards such a goal of taking advantage of already studied problems.

Our interest lies particularly in systems like Hive or Pig, where their high-level languages, `Hive-QL` and `Pig Latin` respectively, are compiled into a series of MapReduce jobs to be executed in a cluster of computers. This is due to the fact that the Hadoop framework needs to be finely tuned, and research works such as [90, 102] provide a propitious level of abstraction where the end user would not have to worry about this fine tunning. In this manner, developers' productivity, and code manageability can be greatly improved. Moreover, this gives the possibility for non-specialized users to take advantage of the computation power of MapReduce on top of their compute clusters. Therefore, our goal is to try to optimize the way these higher-level systems work to make them even more easy to use for users.

There are other systems similar to Pig which also try to reduce the amount of code written and improve code readability e.g. Dryad-Linq[111], Sawzall [43]. But all of them still lack of some traditional database optimizations such as early filtering, projection, operator rewriting, among others. This is due to the fact that, in order to accomplish them, certain characteristics about the data must be known a priori, and that is still a problem if we are dealing with a vast amount of raw data. For instance, when dealing with files stored in the HDFS [33] there is no any additional information in order to enable logical optimizations like in database systems. The information needed to enable optimizations studied in traditional databases goes from simple cardinality to more complex data statistics such as equi-depth histograms [94].

Even though there is work being done towards getting this type of information, they are still experimental. For example, Morton et al. [85] describe how a time-oriented progress indicator can be designed for parallel queries, more specifically for MapReduce directed acyclic graphs (DAGs). The authors do not only take into account the query plans and the amount of data being processed, but also the amount of parallelism available in the cluster in order to estimate the amount of time remaining for a query to complete. Although this approach performs well for estimating the query completion times online, there is no information available about query execution before it is actually executed.

An interesting approach to try to solve this limitation is the work presented in [38, 37]. In this PhD dissertation, Ganapathi [37] describes the use of statistical machine learning in order to obtain the relationships between different input parameters (e.g. number of mappers, number of reducers, input size) with some output parameters (e.g. number of bytes written either into HDFS or locally) to enable the prediction of certain output parameters. Then, Ganapathi et al. [38] prove that their prediction method is suitable for modeling workloads for cloud-like environments where prediction mechanisms are needed to guide scheduling and resource management decisions, and to create realistic workloads to evaluate the choice of policies before using them in production environments.

III.2 Problem Statement

The Pig system allows the use of four different types of join operators (merge, hash, fragment-replicated, and skew joins), but to use them, the user has to pass a hint in Pig-Latin to the optimizer specifying the type of join to be used. This type of communication with the rule-based optimizer might be suitable for some experienced users who probably can choose wisely between the four operators. However, for unexperienced users the constraints and advantages of each type of join operator may not be so clear. In this way, users can be mistaken when choosing the correct type of join operator which can lead to terrible job's performance and misuse of computational resources.

The objective of this dissertation is to investigate the use of statistical methods in order to determine a suitable one for the system architecture presented by Figure III.1. In addition to that, to investigate the parameters involved in creating a statistical model for predicting job ex, more specifically, the design decisions to create a model constructor to provide execution times for the different types of join operators inside the Pig system. This system lacks of a cost model for creating query execution plans. It just translates

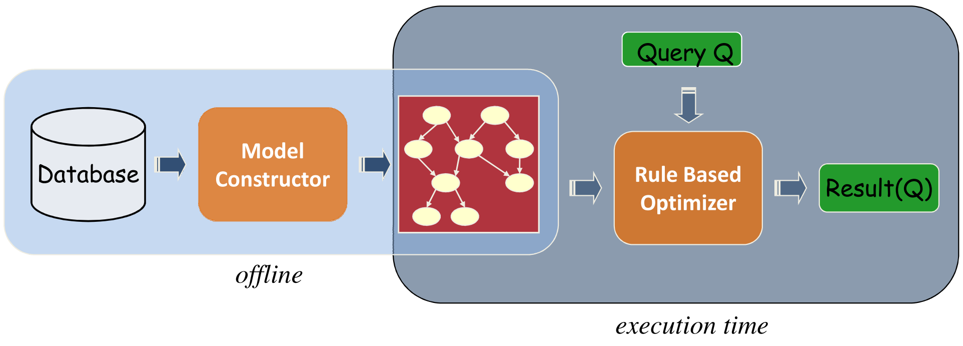

Figure III.1: System Arquitecture

high-level queries into `MapReduce` jobs with a rule-based optimizer. The model to be constructed would help the rule-based optimizer to choose the most appropriate join operator for each situation without letting the user worry about it.

Figure III.1 shows how the system would work after the statistical model has been integrated with it. The Pig-system would receive a join query without any user hint and it would be to able determine the best join operator for the operation. This decision would be based on parameters studied in this dissertation and these parameters would be used by the statistical model previously constructed.

Moreover, we consider this a opportunity see a possibility of applying studied concepts from the database field into large scale data processing. More specifically, we see a chance in integrating these prediction capabilities into the Pig system [31] to improve its query execution capabilities. Figure III.2 shows how integrating these concepts into the system would be like. The left part of the figure exhibits the model constructor of the system. This component should use only *a priori* information about the input relations and the algorithm to create a model that characterizes those data operations specified in the workload. So, we can use the statistical model to obtain query execution times estimates beforehand. The right part of the figure shows how this a priori information can be used by the rule-based optimizer in order to make better decisions about execution plans.

So once the model has been created, and incorporated into the Pig system's rule-based optimizer, every time a join query is submitted, the optimizer would calculate a query execution time estimate to choose the best

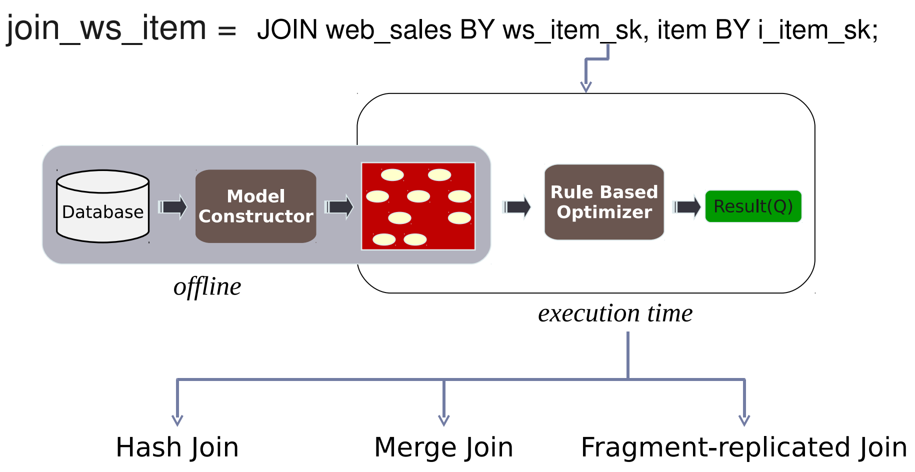

Figure III.2: System behavior for determining the correct type of join.

join method among those available. The actual integration of the model into Pig system's rule-based optimizer is part of our future work. Here, we are just building the first part of our envisioned system.

III.3 Motivation

In this section, we outline the different research works that have motivated this dissertation. Our main goal is to investigate possible ways to construct a model generator for predicting joins operation execution times. This task has been widely studied both in distributed computing and the database system areas.

An interesting research work is the one done by W. Smith [98]. The author explicates the construction of system prediction services. Such a system was built using an instance-based learning technique to predict job execution times, batch scheduling queue wait times and file transfer times of the *Texas Advanced Computing Center Lonestar System* [103]. The prediction error obtained in [98] ranges from 37% to 115%. Even though these are relatively high prediction errors for the training data, the authors discuss that it would be possible to reduce the prediction error by using a larger training dataset. However, they observe that the system on which they are working might be simply less predictable than other systems where similar approaches performed better. Nevertheless, the results obtained show that this approach has lower prediction error in more structured and better behaved workloads when compared to the work discussed in [105].

Estimating query execution time is also a very interesting research topic for the database community. Not only this information can help improving the whole system's performance, but also, it can lead to more proactive decisions of the DBMS (e.g. maintaining statistics, choosing better execution plans). Despite of the similarities of the database world with MapReduce environments, they are still different in many aspects. A very important one is the amount of metadata kept by databases. MapReduce environments tend to keep only operational information (i.e. information they need to keep functioning) in order to remain simple, but reliable.

A known scenario with similar features to MapReduce environments is the one studied by MultiDatabase Management Systems (MDBS) [14]. A MDBS is a distributed system that acts as a front end to multiple local DBMSs. It is structured as a global system layer on top of the local DMBS in order to integrate access to autonomous and heterogeneous local databases. In spite of the fact that the local processing node maintains some global functions to interact with the whole system, the local DBMS participates in the multidatabase without modification [14]. Consequently, the global system does not have access to certain local information such as relation structures, statistics and cost functions. This is the reason why query optimization in an MDBS is also a very interesting research topic.

Another motivating research work is the one done by Zhu et al. [112]. The authors discuss the need of good estimates of selectivities before query execution so that the global query optimizer in an MDBS can choose a low cost execution plan from many alternatives. The author also presents an integrated method to estimate selectivities in an MDBS. He accomplishes this by applying the techniques proposed by Lipton et al. [79] to a MDBS scenario overcoming difficulties such as not knowing the local structure of a relation or not being able to modify it. In [113], Zhu et al. proposed a query sampling method to derive cost estimation formulas for autonomous local database systems in an MDBS. The main idea of their work is to classify local queries with similar costs, sample each query class and use the observed costs to statistically derive local cost estimation formulas for the queries performed on the local databases. If the available information were not enough, they would classify queries with similar performance behavior into the same class so that their estimation errors would not be large.

Recently, different approaches trying to combine more traditional methods with the MapReduce paradigm have been developed in order to improve it. There are some tools/frameworks that work on the MapReduce paradigm or in variation of it. For instance Pig [90], and HIVE [102] work on top of Hadoop [30] which is an open source implementation of the MapReduce framework. These tools try to alleviate the difficulty of having to write MapReduce functions from scratch. They manage to accomplish this by helping developers to express algorithms using the MapReduce model in an easier and more understandable manner. In spite of the fact that a higher-level dataflow language is offered with them, the user still needs to give some hints to the backend components of these tools so they can perform efficiently.

The Pig framework [31] also needs to receive user hints in order to enable its compiler to choose correctly between the different types of join it has implemented. This is clearly not a good way of dealing with the problem because the user would need to know how each type of join works, and even more importantly, because the user could choose wrong hints which in turn would affect the overall performance. Thereby, creating a statistical model to characterize the different types of joins in order to know its execution time beforehand would mean that the most efficient join operator could be chosen automatically and used without any user hint.

Likewise Olston et al. [89] describe the needs in high-level dataflow languages built on top of large-scale parallel dataflow systems. Due to the fact that they provide a faster program development and an easier way to maintain the code, they also bring new difficulties and opportunities for such systems e.g. automatic optimization, query rewriting. The authors argue about the different possible types of optimizations in Data-Intensive Scalable Computing (`DISC`) [15] and in Database Management Systems. In their work, they also point out the differences that make these two contexts similar, but at the same time different.

For example in the DBMS area, highly declarative languages, normalized data and strong consistency are characteristics usually taken for granted. By contrast, `DISC` uses more procedural code with flexible data models and cost-effective scalability using weak consistency and commodity hardware. Another important aspect addressed by the authors is the difference between the optimizations techniques between traditional DBMS and `DISC` applications. The former tries to get the best execution strategy over the data models, operators and execution environment whereas the latter may not have accurate models *a priori*.

The lack of data models is due to problems such as:

– Data usually residing in plain files and the user has to instruct the system on how to parse them.

– Many of the operators are custom user-supplied code whose cost is not known *a priori*.

– DISC applications may use large pools of unreliable and probably heterogeneous machines.

As a consequence of these problems, formulating simple and accurate models of the execution environment is still a challenge.

High-level dataflow systems that use the MapReduce paradigm could benefit from the use of cost estimates. Those estimates should not have a negative impact on the system performance e.g. longer execution times and extra complexity. They should be obtained using only known information such as cluster capacity, input size, among other known execution parameters.

Therefore, we would like to explore different approaches to extract the relationships between input parameters and measured performance. Using statistical methods to identify these relationships, we could extrapolate "*what-if*' workload scenarios using the statistical model created as a functional instance for the whole system. Our interest lies specifically on join operation because it is generally the most time-consuming and it is still user-driven in the Pig system.

In this chapter we have described how this thesis relates with the current large scale data management trend. Besides that, we have explained the problem this dissertation targets and the research works that have motivated our research. In chapter V, we will describe other research initiatives with similar goals as our.