V
Conclusions

V.1 Related work

Even though MapReduce appears to be constructed specifically for performing group-by aggregations, there are also many interesting research work being done on studying critical data processing operations such as joining sets of data. In this section we will describe some of these related works.

Blanas et. al. [13] presented an experimental performance evaluation of different implementation of join strategies using the MapReduce framework, but also suggested algorithms for joining log datasets with user datasets. The authors also compared their approach with Yahoo's Pig Project [31] and showed that their proposed join algorithms performed better than those implemented in Pig. However, the comparison was done with a relatively old version of Pig (version 0.2). Pig's current version is 0.8.1 and it already contains most of the improvements proposed by [13]. We consider that the most important contribution made by Blanas et. al. is the decision tree which summarizes the tradeoffs of various join strategies in the MapReduce framework. Such decision tree could be used as an initial set of rules for choosing the right join operator. We consider this specific contribution one of the most important made in this research path because most of the work done try to prove such proposal.

A more theorical research work is the one made by Palla [91] where a theoretical cost model to evaluate the I/O cost incurred was developed. Besides that, the author applied the proposed cost model with two popular join algorithms using MapReduce, and to a join variation also proposed. A difference with such work is that in their work basic statistics are assumed as given such as cardinality, and also join selectivity is not considered in the cost model proposed. The author uses the input size, the bytes read from the network, the bytes written to the network (HDFS), among other parameters to determine the I/O cost of MapReduce job in their different phases i.e. map, merge and reduce phases.

An extension to Palla's work is the research done by Jairam Chardan in [19]. Chardan proposed an evaluation of the different types of multiway joins and proposed improvements for the hash join operator, a variation of the hash join called "Reduce-side cascade join" in which a regular hash join is performed for every two pair of relations but also deciding the order of joining datasets in a pre-processing step. The approach taken by Chardan shows that getting basics statistics is still a work in progress in the MapReduce context because pre-processing the relations might be a burden still to be evaluated.

Multiway joins are another interesting problem when processing large amounts of data as well as in web or data warehousing applications. In that way, Afrati et. al. [2] show how the MapReduce framework identifies which value will go to a specific reducer in order to optimize the shares (number of buckets into which the values of the map-key are hashed) when having a fixed number of reducers. Thus, they propose an approach to identify the map-key i.e. the set of attributes that identify the reducer to which each mapper must send a particular record. They consider two important special cases: chain joins and star joins. They explore how to choose the map-key and shares to minimize the communication cost using the method of "*Lagrangean multipliers*" to set up the communication-cost-optimization. The authors consider that their solution for optimizing multiway joins should be viewed as a suggestion rather than a requirement because their method is not always superior to the conventional way of implementing joins in MapReduce.

Another interesting research path similar to the one proposed by our work is determining job execution times. This task has been widely studied in distributed computing e.g. [98, 112, 105, 99], among many other research work using different methods to accomplish it.

For example, the work by W. Smith in [98] explicates the construction of a system which uses 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]. Due to the relatively high prediction error for the training data, the authors proposed get a bigger training dataset used to reduce the prediction error. In spite of these improvement possibilities, the authors remark that their system might be less predictable than others where similar approaches performed better. However, the results obtained show that their approach has lower prediction error in more structured and better behaved workloads as compared to the work discussed in [105].

On the other hand, estimating query execution time is also a very interesting research topic for the database community. Such information could

help improving more proactive decisions of the DBMS (maintaining statistics, choosing better execution plans, etc.). In spite of the similarities of the database world with MapReduce environments, they are still different in many aspects. A noticeable one is metadata management by databases whereas MapReduce environments tend to keep only operational information i.e. information they need to keep functioning. However, heavy research is being carried on by the academia, and also by the industry in order to improve such environments.

Multi Database Management Systems (MDBS) has similar features to MapReduce environments. A MDBS is a distributed system that acts as a front end to multiple local DBMSS or is structured as a global system layer on top of the local DMBS to integrate autonomous, heterogeneous local databases. The local node maintains some global functions to interact with the global system, and participates in the multidatabase without modification [14]. In this manner, the global system does not have access to certain local information such as local relation structures, local statistics, local cost functions, among other types of information. This is the reason why query optimization in an MDBS is also a very interesting research topic.

For instance, Zhu [112] discusses the need of good estimates of selectivities before execution of the queries in order to 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 based on the discussion. He accomplishes this by applying the techniques proposed by Lipton et al. [79] to an MDBS scenario overcoming difficulties such as not knowing the local structure of a relation, or not being able to modify it. In addition to that, in [113], Zhu et al. proposed a query sampling method to derive local 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 to use the observed costs of the sample queries and statistically derive local cost estimation formulas for the queries performed on the local databases i.e. using query sampling for estimating local query costs in an MDBS. 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. An interesting work is [37]. The author, A. Ganapathi, shows also that statistical methods can be applied in order to determine query completion times for parallel databases, but also for MapReduce jobs. The author applied statistical machine learning to predict not only job execution times, but also

the number of bytes read and written locally, bytes read and written from the network by the MapReduce functions. In addition to that, Ganapathi et al. [38] argue how these predictions before query execution instead of halfway through the execution is a clear advantage in cloud-like environments. This is because that these statistical models could be used to predict resource requirements for data-intensive workloads in specialized environments, and help improve job scheduling, workloads generation, among other opportunities. The research done in [37, 38] show that statistical machine learning is a powerful tool for estimating different performance parameters using only a-priori information.

A similar work is the one presented by Kavulya et al. in [72] where the statistical analysis of logs from a production MapReduce cluster is analyzed. The authors explicate the statistical analysis done of a production MapReduce cluster trace log from `Yahoo!`. This analysis was performed using statistical tools such as distance-weighted average, and locally-weighted linear regression.

Although [37, 38] present lower error metrics, Kavulya et al. achieve a good statistical characterization of the jobs executed by analyzing 10-months of trace data from the M45 [65] supercomputing cluster. Other important findings of Kavulya et. al are that the variability in user behavior over short periods of time was low, and that average resource utilization on the cluster was relatively low e.g. CPU utilization across all nodes ranged from 5% to 10%.

An important conclusion from [37, 72, 38] is that these statistical methods are useful as long as the training set is large enough to successfully construct the model. The work presented by Ganapathi [37] is based on the production logs from `Facebook`, and the work presented by Kavulya [72] uses the production logs from `Yahoo!`.

Other approaches studied how to estimate completion times [86, 85]. In [86], the authors presented a progress estimator for MapReduce pipelines called Parallax. The biggest contribution was its estimation per-tuple processing costs, but also taking into consideration cluster capacity and the (estimated) dataset sizes. Their progress estimator works by calculating the remaining time for a sequence of pipelines as the sum of the time-remaining for the pipelines being currently executed, but also for the ones to be executed. It breaks queries into pipelines i.e. groups of interconnected operators that would execute at the same time considering only the times from the record reader, map runner, and combiner operations taken together, the copy and the reducer. Another important aspect is that their initial work did not take into account was the sorting time which was considered by their complementary work [85]. Besides that, Parallax progress estimator contributes in the way it

handles parallelism i.e. multiple nodes processing a map or reduce task at the same time.

The research work presented in [85] is an extension of Parallax which was called ParaTimer. It builds on Parallax but takes a radically different strategy for progress estimation because it adopts a critical-path-based progress estimation technique. ParaTimer recognizes and monitors only the map and reduce tasks that are part of the query's critical path. Another idea introduced by Paratimer is to provide users with a set of estimated query runtimes.

At the same time, there are many initiatives in the open-source community with the projects that started from academic publications. For instance, the Hive project [34] has many collaborators around the world trying to improve the system. In spite of its current optimization being rule-based (i.e. involves applying a set of rules on the *Plan tree*), it depends on hints given by the user. These hints may or may-not be correct which could result in execution of costlier plans.

That is one of the reason why tasks such as [51, 50], and others have been started within the Hive project. The task described in [51] aims at building a cost-model which can give a good estimate various plans before hand (using some meta-data already collected), so the system can choose the best plan incurring in the least cost. This task was created in February of this year. In addition to that there are subtasks such as [50, 48], among others. [50] is called hive-1362 and its objective is to gather column level statistics to help improve the rule-based optimizer whereas [48] is called hive-33 and aims to add the ability to compute statistics on hive tables.

These initiatives are similar in spirit to our work in which they try to improve the rule-based optimizer used in the Hive system. We believe that in the future some basic statistics must be gathered to make more accurate estimates about data processing tasks at internet scale. Then, when basic statistics about the data have been obtained, other different models could be used to predict query execution times, query output size, join selectivity, or other useful values for data management systems.

V.2 Conclusions

In this work a methodology for statistical modeling of the MapReduce join operator has been presented. This approach allowed us to explore different aspects of the problem e.g. determining model parameters, getting join selectivity with very little knowledge of the relations, or even getting simple database statistics such as record average length, or cardinality of the relations, and doing all this without perishing viability of the model construction.

In fact, the use of statistical tools to model MapReduce jobs has drawn the attention of the research community in the last years.

The main motivation for this work lies in the observations pointed out by works such as [27, 78, 37, 38, 72]. Ganapathi [37] shows how statistical machine learning can be applied to predict not only job execution times, but also number of bytes read and written locally for Map and Reduce functions. Kavulya et al. [72] explicate the statistical analysis done of a production MapReduce cluster trace log. This analysis was performed using statistical tools such as distance-weighted average, and locally-weighted linear regression. Although Ganapathi et al. [37, 38] presented lower error metrics, Kavulya et al. achieve a good statistical characterization of the jobs executed by analyzing 10-months of trace data from the M45 [65] supercomputing cluster. Ganapathi et al. [38] also state that being able to make predictions before query execution instead of halfway through the execution is a clear advantage in cloud-like environments. This is due to the fact that these statistical models could be used to predict resource requirements for data-intensive workloads in such environments. On the other hand, the idea of characterizing a relational operator comes from works such as [27, 78] where the authors demonstrate that user behavior is uniform and that is also predictable in short periods of time. This makes the possibility of characterizing relational operators in the absence of a cost-based optimizer, an interesting possibility, at least until an efficient way to gather statistics in such distributed environments become available.

Despite the fact that sampling methods perform well in traditional environments, they did not performed well in a MapReduce environment. We hypothesize this is because of two reasons. The first one is the amount of data needed to make the sampling method to more accurate might incur in excessive costs due to the scale MapReduce has been designed for. And the second reason is that the error obtained while getting basics statistics has an important impact when join selectivity is calculated. These basic statistics errors influence the error of the join selectivity parameter which in turn does not greatly contribute to the regression model.

As pointed out by Kavulya et al. [72], some input features were linearly dependent in certain jobs. In our specific case, it was the number of reducers used. It is because this depends on cluster's capacity if not set specifically to a lower number, and our experiments were done on a small local infrastructure without specifically setting it. This collinearity problem can cause the model construction process to fail due to the fact that linear regression in the presence of collinearity can produce unstable estimates. It would be interesting to vary this parameter according using some a priori knowledge of the distribution of

the data.

We also noticed that large error-latencies in some long-running tasks indicate that better diagnosis and recovery approaches are needed. Some of these long-running tasks were tasks that failed many times before they totally failed or finally finished executing. It would be interesting to also predict job failures, and how different recovery approaches could be used in such situations.

Due to the fact that our dependent variable is the query execution time, scaling the input data would not affect the model itself but would only vary the scale of the execution times obtained i.e. the queries would take longer to be executed, but the relationships among them would remain the same.

A major problem encountered while developing a predictive model is that the data set available for analysis is relatively small, and our second experiment helped us perceive that our model was being overfit i.e. the estimates obtained were the result of poor performance at predicting values, and the models created were probably misrepresenting small variations in the data. In this way, we believe that a bigger dataset or other methods to alleviate the overfitting problem should be studied as future work.

V.3 Summary of Contributions

Our contributions put in a shot way are:

- *Estimating join selectivity in a MapReduce environment.* We adapted a sampling-based method to estimate join selectivity in queries in order to use it as another parameter for modeling query execution times. We decided to use this method because it makes no assumptions about the underlying data and does not require storing and maintaining detailed statistics. These characteristics make sampling an interesting option for obtaining information about the data without actually incurring in high performance costs.
- *Building a statistical model for query execution time prediction.* We built a multiple regression model in order to predict the execution time of join queries. In this way, we wanted to characterize the different types of join implemented in the Pig Framework, specifically fragment-replicate join, merge join and hash joins. We also tried adding the join selectivity parameter into the models because it is a very relevant parameter for determining output size, and therefore constructing a better characterization of the join operator.
- *Constructing a dataset of join queries.* One of the biggest difficulties found while working on this research project was the design and con-

struction of the dataset. This is due to the fact that there is not an available data set to work on which our specific requirements would have been fulfilled i.e. queries running join operations on a MapReduce environment. In spite of the fact that there are datasets available on which data intensive analysis can be performed, there are not a set of specific queries testing particular parts of systems working on MapReduce. For our particular case, the only set of queries is the twelve queries run on top of the Pig framework to test correctness and performance from release to release [36]. Therefore, we decided to construct our own dataset to produce a higher number of queries for each type of join type, and so then we could be able to create a richer model i.e. greater variety of queries allowed us to do a more realistic study of the system.

– *Using the statistical model to characterize the join operator in MapReduce.* We used the model built with the join queries to attempt to characterize three types of join operators implemented in the Pig framework. We created a model for each type of join, and evaluated them using a three-fold validation. The results obtained present interesting insides about the join operation in MaprReduce e.g. the hash join operator depends heavily on cluster resources because it needs main memory in order to probe relations; the merge join operator seems to adequate easily for such modeling, but data distribution has to be taken into consideration because query performance can vary depending on it; and the fragment replicate join also depends heavily on the cluster resources as it uses the distributed cash to send data to the nodes where it will be probed, so in such jobs, hardware consideration should be taken into consideration. These hardware considerations should be done in a similar way to related work such as [98, 105, 99, 37].

V.4 Future Work

Even though the number of queries created and used as our data set was not too small, we perceived through our second experiment that the number of queries executed needs to be expanded, and so the variety within them. Having a relatively small data set led us to overfit our model. Therefore, one of the solutions to this problem is to create a bigger dataset, or to study some different approaches to avoid overfitting our regression model e.g. shrinkage methods [95]. These methods make more modest prediction which are closer to the average in order to reduce overfitting the model.

A solution to create a bigger dataset could be to use a different dataset from the database world to create more queries for each type of join operator. This would mean add more variety of job executions to the model. We think that this would be an interesting idea because we could choose a dataset not designed for `OLAP` queries like the `TPC-DS`, but we could use one designed for `OLTP` queries such as the `TPC-C`, or the `TPC-WS`.

In addition to that, we believe that exploring other statistical methods as the ones proposed by [98, 72, 37] are very appealing to the approach presented in this work. This is because kernel methods used in [37] show great advantages compared to regular statistical methods. In spite of that, locally weighted learning [72] and instance based learning [98] seem suitable and simple approaches for our problem. It would be interesting to compare such methods in order to determine the best of them in terms of performance, accuracy, and complexity to add them to MapReduce systems.

An interesting observation from our results is that in spite of the advantages of sampling methods to determine selectivities, these might not be accurate without having to sample too many tuples from the relations i.e. without increasing the cost of performing them. That is why we believe that other methods to obtain join selectivity from queries should be also studied. For instance, Getoor et. al. [40] propose to utilize probabilistic graphical models, which takes advantage of conditional independence relations between the attributes in the table to allow a compact representation of the joint distribution of the attribute values involved. The authors' approach is to consider the joint distribution over multiple attributes, and not just the distributions over the attributes in isolation, and not to follow common assumptions of DMBS e.g. attribute value independence assumption, join uniformity assumption.

In the other hand, there are many initiatives in research groups from Yahoo!, Facebook, StumpleUpon, and others, that aim to improve the performance of systems that work on top of Hadoop. Such research initiatives are nowadays projects supported by the `Apache Software Foundation` [32] e.g. Pig [31], HIVE [34], Howl [57], HBase [101], among others projects. In this manner, we consider that getting involved with such development groups is an opportunity to get better insights of the systems, but also to be able to contribute to research in a broader aspect.

Therefore, once the chosen model is able to make accurate predictions about query execution times, we would be able to add new optimization rules into the Pig Framework. In this way, we would be alleviating the lack of a cost-based optimizer while research on obtaining and maintaining statistics

is currently being done, and at the same time contributing to open source projects.