ReStore: Reusing Results of MapReduce Jobs in Pig

Iman Elghandour
University of Waterloo
ielghand@cs.uwaterloo.ca

Ashraf Aboulnaga
University of Waterloo
ashraf@cs.uwaterloo.ca

ABSTRACT

Analyzing large scale data has become an important activity for many organizations, and is now facilitated by the MapReduce programming and execution model and its implementations, most notably Hadoop. Query languages such as Pig Latin, Hive, and Jaql make it simpler for users to express complex analysis tasks, and the compilers of these languages translate these complex tasks into workflows of MapReduce jobs. Each job in these workflows reads its input from the distributed file system used by the MapReduce system (e.g., HDFS in the case of Hadoop) and produces output that is stored in this distributed file system. This output is then read as input by the next job in the workflow. The current practice is to delete these intermediate results from the distributed file system at the end of executing the workflow. It would be more useful if these intermediate results can be stored and reused in future workflows. We demonstrate ReStore, an extension to Pig that enables it to manage storage and reuse of intermediate results of the MapReduce workflows executed in the Pig data analysis system. ReStore matches input workflows of MapReduce jobs with previously executed jobs and rewrites these workflows to reuse the stored results of the matched jobs. ReStore also creates additional reuse opportunities by materializing and reserving the output of query execution operators that are executed within a MapReduce job. In this demonstration we showcase the MapReduce jobs and sub-jobs recommended by ReStore for a given Pig query, the rewriting of input queries to reuse stored intermediate results, and a what-if analysis of the effectiveness of reusing stored outputs of previously executed jobs.

Categories and Subject Descriptors

H.2 [**Database Management**]: Miscellaneous

Keywords

MapReduce, Data Reuse

SIGMOD '12, May 20–24, 2012, Scottsdale, Arizona, USA.

1. INTRODUCTION

Massive scale data analysis has become a required activity for many enterprises. Companies such as Facebook, Yahoo, and Google now own petabyte-scale data warehouses that are accessed using ad hoc queries and periodic batch jobs, and terabyte-scale data warehouses are now common in many smaller organizations. This large scale data analysis is currently supported by the MapReduce programming and execution model [3] and its implementations such as Hadoop [1]. However, the data analysis tasks performed by Hadoop users are usually complex and require high-level query languages such as Pig Latin [5] to express them. The compilers of these query languages translate queries into *workflows of MapReduce jobs*. Each MapReduce job implements multiple *physical operators* such as *Load*, *Filter*, and *Join*. A DAG of these operators comprises the physical query execution plan of a job. The output of each MapReduce job in a workflow is stored in the distributed file system used by the MapReduce system, and is consumed as input by the next job in the workflow. When the final result of the workflow is produced, these intermediate results are deleted.

In this demonstration, we present ReStore [4], a system that enables the Pig data analysis system to reserve (i.e., store) intermediate results generated during workflow execution instead of deleting them, and to reuse these stored results in future workflows executed by the system. This result reuse leads to significant performance improvements for Pig Latin queries, which we showcase during the demonstration. ReStore extends Pig to achieve two goals: (1) rewriting workflows of MapReduce jobs to reuse any past results reserved by ReStore and (2) maximizing the reuse opportunities between MapReduce jobs that are independently executed in the system. It is important to note that matching and sub-job enumeration are based on physical plans. Even though we developed ReStore as an extension to Pig, the techniques that it uses can be applied in any dataflow system that generates workflows of MapReduce jobs for input queries such as Hive and Jaql.

In the next section, we present a high-level overview of ReStore and describe its components that extend Pig. More details are available in [4]. In Section 3, we describe our demonstration scenario, which showcases the details of preparing a Pig query to be executed, with a special focus on: (1) rewriting the workflow of MapReduce jobs created by Pig for an input query to reuse outputs of past jobs reserved by ReStore, (2) generating additional reuse opportunities in ReStore, and (3) the tools provided to analyze the performance of the queries that are rewritten by ReStore.

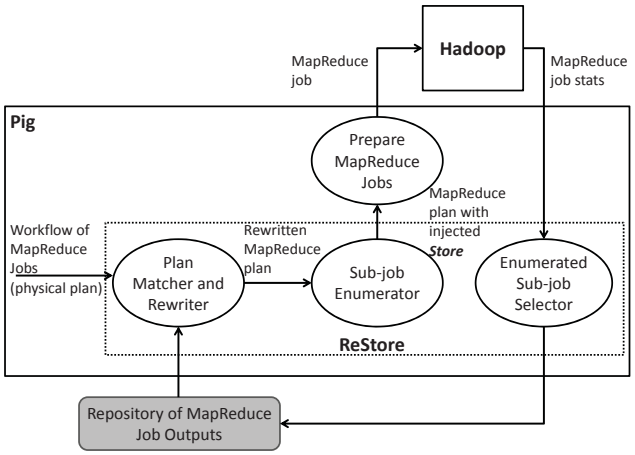

Figure 1: ReStore system architecture.

2. THE RESTORE SYSTEM

Pig [5] is a dataflow system that accepts queries in a SQL-like query language called Pig Latin, and it compiles these queries into workflows of MapReduce jobs that are executed on Hadoop. The Hadoop job manager of Pig is responsible of submitting the MapReduce jobs in the compiled workflow to Hadoop for execution, taking into account the dependencies between them. One of the Hadoop job manager components is the JobControlCompiler that takes as input the compiled workflow of MapReduce jobs and iterates though it to decide on the jobs that can be run concurrently. Every iteration, the JobControlCompiler creates MapReduce Java objects from the physical plans of the selected jobs and submits these objects to Hadoop for execution. The outputs of the MapReduce jobs that are used as input to other jobs in the workflow are stored in the Hadoop Distributed File System (HDFS). After executing all the MapReduce jobs in the workflow, these intermediate results are deleted.

ReStore (Figure 1) extends the JobControlCompiler component of Pig. The input to ReStore is a workflow of MapReduce jobs generated by Pig for an input query. The outputs of ReStore are: (1) a modified MapReduce workflow that exploits the results of past jobs executed by Hadoop and stored by ReStore, and (2) a new set of MapReduce job outputs to store in HDFS. ReStore keeps a repository of metadata for the reserved job outputs, in which it stores for each reserved job output: (1) the physical plan of the MapReduce job that was executed to produce this output, which contains information about the input data, the output data, and the operators that were executed to compute this output data, (2) the filename of the output in HDFS, and (3) statistics about the cost of the MapReduce job and the frequency of use of its output by different workflows.

ReStore has three main components: (1) plan matcher and rewriter, (2) sub-job enumerator, and (3) enumerated sub-job selector. For each job in the input workflow of MapReduce jobs, the plan matcher and rewriter searches the ReStore repository for results of past jobs that can partially or fully compute the answer to the input job, and rewrites the input MapReduce job to reuse any results that it finds. Hence, we exploit any opportunity to reuse results of past jobs. The second component of ReStore, the sub-job enumerator, enumerates for a given input MapReduce job the subsets of the physical operators within this job (DAGs of physical operators, or sub-jobs) that can be materialized and stored in the repository. After executing the rewritten job in the MapReduce system (i.e., Hadoop), the enumerated sub-job selector, which is the third component of ReStore, examines statistics collected during job execution about the run time and input/output data sizes of the MapReduce job, and uses these statistics to select some of the enumerated sub-jobs for reserving their output in the repository. Next, we discuss these three components of ReStore in more detail.

2.1 Matching Input MapReduce Jobs with Plans from the Repository

The first phase of ReStore is the plan matcher and rewriter, which works on the physical plan of the input workflow of MapReduce jobs. Matching and rewriting processes one MapReduce job at a time, and before a job is matched against the repository, all other jobs whose output this job reads as input have to be matched and rewritten to use the job outputs stored in the repository. For every job in an input workflow of MapReduce jobs: (1) the plan matcher checks the physical plans stored in the repository for previously executed MapReduce jobs to decide which of them can best be used to rewrite the physical plan of the input MapReduce job, and (2) the plan rewriter uses the chosen plans from the repository to rewrite the job.

For each input MapReduce job, ReStore scans sequentially through the physical plans in the repository and tests whether each plan matches the input MapReduce job. It is possible that the physical plan in the repository matches the entire input MapReduce job, and therefore we rewrite the other MapReduce jobs in the workflow that use the output of this job to load the data from the output of the matched job stored in the repository. The other possibility is that the physical plan in the repository matches part of the input MapReduce job, and therefore the input MapReduce job is rewritten to use the output of the matched job stored in the repository, and ReStore continues matching the rewritten plan with plans in the repository.

ReStore implements an algorithm that is based on operator equivalence to test whether a physical plan in the repository is contained in the physical plan of the input MapReduce job. Two operators are equivalent if: (1) their inputs are read from operators that are equivalent or from the same data files, and (2) they perform functions that produce the same output data. To match two physical plans, both plans are traversed simultaneously starting from the *Load* operators, which read the data from HDFS files, and recursively finding matching operators until either the full plans are matched or the first mismatching operator is found. Several physical plans from the repository can be potential matches to any input physical plan, and multiple plans from the repository can be used together to rewrite the physical plan of the input MapReduce job. Every time ReStore matches an input MapReduce job with the plans stored in the repository, it uses the first match that it finds in the repository to rewrite the input job. This makes matching more efficient, but requires us to order the physical plans in the repository so that the first match found is the best match (i.e., the one that achieves the maximum reduction in the execution time of the input workflow) [4].

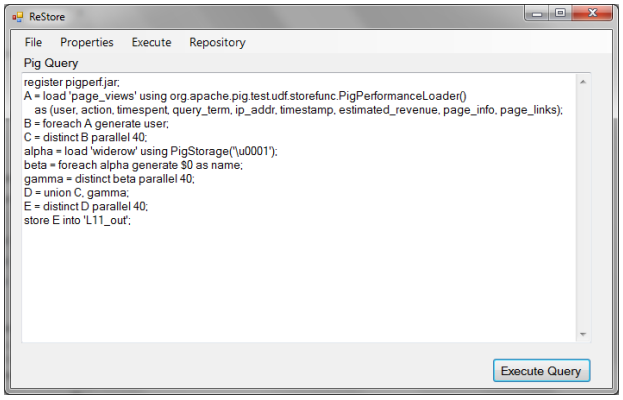

Figure 2: Input Pig query.

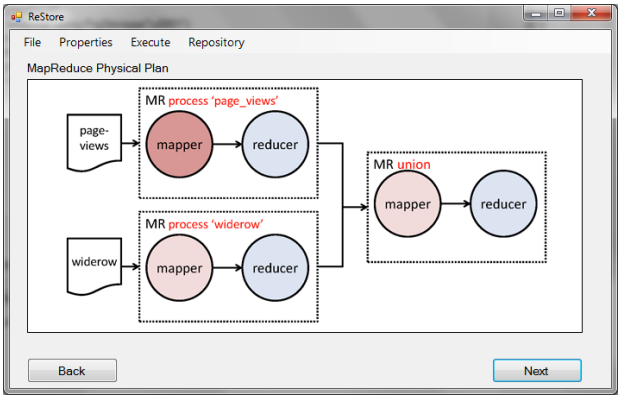

Figure 3: Workflow of MapReduce jobs generated by Pig.

2.2 Generating New Reuse Opportunities

The output of a whole MapReduce job may be useful for future workflows that are executed in the system. This output can be reserved for free because it is already stored in HDFS, and therefore, it is a candidate for reservation in ReStore. However, it may not be easy to find a complete MapReduce job occurring in multiple workflows. Each MapReduce job in Pig and similar systems implements a sequence of physical operators, and it is more likely to find a sequence of physical operators that forms part of a MapReduce job in one workflow occurring again in a different MapReduce job in another workflow. To take advantage of this reuse opportunity, ReStore has the ability to reserve the outputs of individual physical operators within a MapReduce job, which we refer to as outputs of *sub-jobs*. ReStore can materialize the output of a sequence of physical operators in a MapReduce job and store it in HDFS.

The technique we use to enumerate candidate sub-jobs is as follows. We parse the physical plan of a MapReduce job starting from its *Load* operators. After every parsed operator in the physical plan, we inject a new *Store* operator if the parsed operator is not already a *Store* or followed by a *Store*. The *Store* operator is a Pig operator that stores its input in HDFS. However, to include this *Store* operator in the physical plan of the MapReduce job we need to also insert an operator that splits the data into two branches, similar to a Unix `tee` command. An example of this branching operator is the *Split* operator in Pig. The output of the last operator in the sequence of physical operators of the candidate sub-job is pipelined into the injected *Split* operator. One branch of the output of the *Split* operator is pipelined into successor operators in the original physical plan, and the other branch is pipelined into the new *Store* operator. Figure 6 shows a MapReduce plan after injecting a *Store* operator after a *Project* operator.

Treating all possible sub-jobs as candidates and storing their outputs in the distributed file system during the execution of the input MapReduce workflow has the following problems: (1) it would require a substantial amount of storage in the distributed file system, and (2) the overhead of storing all this intermediate data would considerably slow down the execution of the input MapReduce job. Furthermore, some of these sub-jobs may not be useful for future workflows. Thus, we need to select only a subset of the possible sub-jobs to consider as candidates. To achieve this we use one of the following two heuristics for choosing candidate sub-jobs: (1) Conservative Heuristic: the outputs of operators that are known to reduce their input size (e.g., *Project* and *Filter*) are used as candidate sub-jobs, and (2) Aggressive Heuristic: the outputs of operators that are known to reduce their input size and also the outputs of operators that are known to be expensive (e.g., *Join* and *Group*) are used as candidate sub-jobs.

2.3 Managing the ReStore Repository

It could be expensive to keep the output of all generated jobs and sub-jobs in the repository in the long run because of the storage space required and the increasing number of plans to match with future workflows. Therefore, after executing a workflow of MapReduce jobs that was augmented with extra *Store* operators by the sub-job enumerator component of ReStore, the enumerated sub-job selector examines the statistics generated by Hadoop during job execution and selects the jobs and sub-jobs whose output to keep in the repository. This decision is made *after* the workflow is executed, so it is possible to base it on accurate execution statistics. The selected job (or sub-job) whose output we keep in the repository is a job that can reduce the execution time of a workflow that contains this job when replaced with a *Load* from an existing data source. In addition, we also evict stored job outputs from the repository that have not been used within a window of time or when one or more of their inputs is deleted or modified.

3. DEMONSTRATION DESCRIPTION

Our extensions to the Pig system allow us to: (1) rewrite input workflows of MapReduce jobs to reuse outputs of jobs reserved by ReStore, and (2) generate new data reuse opportunities. Our demonstration illustrates these two capabilities with a focus on how they affect query compilation and execution in Pig. ReStore also provide tools to analyze the effectiveness of the data reuse opportunities that it generates, and we use these tools to demonstrate the benefit that a user derives from result reuse in ReStore. The demonstration uses data sets and queries from benchmarks such as PigMix [2]. For a given Pig query, ReStore uses the Pig compiler to translate the input query into a workflow of MapReduce jobs. Figures 2 and 3 show an example input Pig query that a user can input in our demonstration and

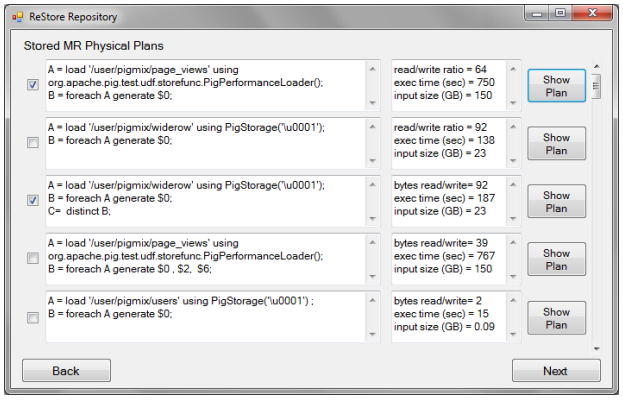

Figure 4: Physical plans stored in the ReStore repository.

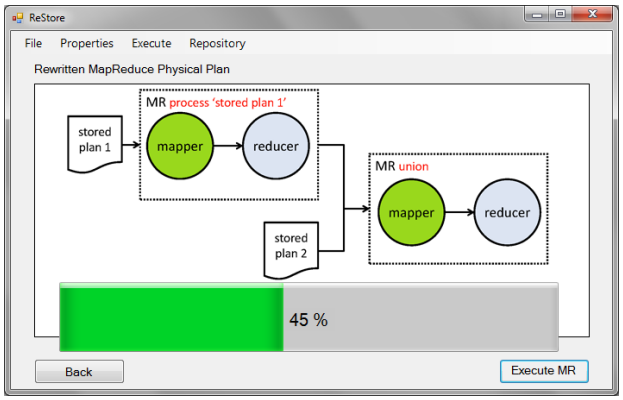

Figure 5: MapReduce workflow of the query after rewriting it using the stored plans.

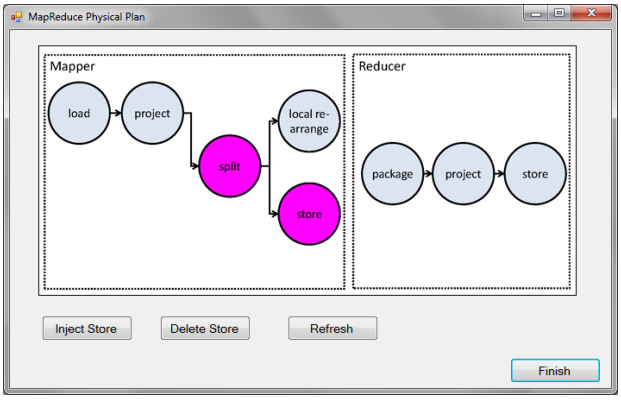

Figure 6: Injecting *Store* operators in the MapReduce physical plan.

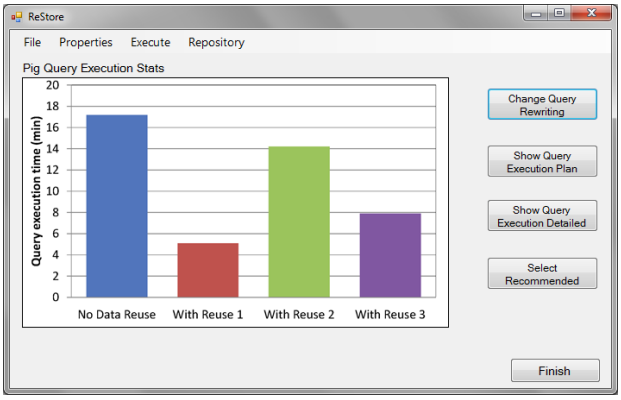

Figure 7: Performance monitoring and analysis in ReStore.

its corresponding workflow of MapReduce jobs generated by Pig. Next, we describe the main functionalities of ReStore that are illustrated in this demonstration and how the user can make changes to ReStore choices to analyze them.

Matching Input MapReduce Jobs with Plans from the Repository. The user of ReStore can browse through the list of physical plans in the repository that can be used for the jobs in the input workflow (Figure 4) and can choose another set of physical plans from the ReStore repository and override the ReStore selected plans. Figure 5 shows the workflow after rewriting it to use the job outputs stored in the repository (the rewritten jobs are shown in green).

Generating New Reuse Opportunities. ReStore injects extra *Store* operators in the plan to reserve the outputs of sub-jobs (Figure 6). The user of ReStore can choose one of the heuristic levels described in Section 2.2 or can manually select the locations in the plan to inject *Store* operators.

Analyzing the Performance of ReStore. After rewriting the workflow of MapReduce jobs to reuse stored data and inject extra *Store* operators in it, the jobs in the workflow are prepared by Pig and submitted to Hadoop for execution (Figure 5). Demo participants will be able to see the execution time and detailed statistics about the execution of the workflow of MapReduce jobs. The system provides the ability to compare the execution time of the original workflow with no reuse and the workflow recommended by ReStore that reuses data stored in the repository. Additional data reuse choices can be made by the user of the system to override the choices that ReStore makes, which will result in alternate workflows. The modified workflows can be executed by the system and compared to previous executions of the input query (Figure 7). Finally the user can view the enumerated sub-jobs that are generated by ReStore and decide on those to keep in the repository and those to delete.

4. ACKNOWLEDGMENTS

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Business Intelligence Network strategic networks grant.

5. REFERENCES

[1] Apache Hadoop. `http://hadoop.apache.org/`.
[2] PigMix. `http://wiki.apache.org/pig/PigMix`.
[3] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. OSDI*, pages 137–150, 2004.
[4] I. Elghandour and A. Aboulnaga. Restore: Reusing results of mapreduce jobs. *Proc. VLDB Endow. (PVLDB)*, 5(6):586–597, 2012.
[5] C. Olston et al. Pig Latin: a not-so-foreign language for data processing. In *Proc. SIGMOD*, pages 1099–1110, 2008.