Information Systems 60 (2016) 69–84

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

FP-Hadoop: Efficient processing of skewed MapReduce jobs

Miguel Liroz-Gistau [a], Reza Akbarinia [a,*], Divyakant Agrawal [b], Patrick Valduriez [a]

[a] INRIA Montpellier, France
[b] Department of Computer Science, University of California, Santa Barbara, United States

ARTICLE INFO

Article history:
Received 10 February 2016
Accepted 23 March 2016
Recommended by: D. Shasha
Available online 1 April 2016

Keywords:
MapReduce
Data skew
Parallel data processing

ABSTRACT

Nowadays, we are witnessing the fast production of very large amount of data, particularly by the users of online systems on the Web. However, processing this big data is very challenging since both space and computational requirements are hard to satisfy. One solution for dealing with such requirements is to take advantage of parallel frameworks, such as MapReduce or Spark, that allow to make powerful computing and storage units on top of ordinary machines. Although these key-based frameworks have been praised for their high scalability and fault tolerance, they show poor performance in the case of data skew. There are important cases where a high percentage of processing in the reduce side ends up being done by only one node.

In this paper, we present *FP-Hadoop*, a Hadoop-based system that renders the reduce side of MapReduce more parallel by efficiently tackling the problem of reduce data skew. FP-Hadoop introduces a new phase, denoted *intermediate reduce* (IR), where blocks of intermediate values are processed by intermediate reduce workers in parallel. With this approach, even when all intermediate values are associated to the same key, the main part of the reducing work can be performed in parallel taking benefit of the computing power of all available workers.

We implemented a prototype of FP-Hadoop, and conducted extensive experiments over synthetic and real datasets. We achieved excellent performance gains compared to native Hadoop, e.g. *more than 10 times in reduce time and 5 times in total execution time*.

1. Introduction

In the past few years, advances in the Web have made it possible for the users of information systems to produce large amount of data. However, processing this big data is very challenging since both space and computational requirements are hard to satisfy. One solution for dealing with such requirements is to take advantage of parallel frameworks, such as MapReduce [1] or its IO-efficient versions such as Spark [2], that allow to make powerful computing and storage units on top of ordinary machines.

The idea behind MapReduce is simple and elegant. Given an input file of key-value pairs, and two functions, map and reduce, each MapReduce job is executed in two main phases. In the first phase, called map, the input data is divided into a set of splits, and each split is processed by a map task in a given worker node. These tasks apply the map function on every key-value pair of their split and generate a set of intermediate pairs. In the second phase, called reduce, all the values of each intermediate key are grouped and assigned to a reduce task. Reduce tasks are also assigned to worker machines and apply the reduce function on the created groups to produce the final results.

* Corresponding author.
E-mail addresses: miguel.liroz_gistau@inria.fr (M. Liroz-Gistau), reza.akbarinia@inria.fr (R. Akbarinia), agrawal@cs.ucsb.edu (D. Agrawal), patrick.valduriez@inria.fr (P. Valduriez).

http://dx.doi.org/10.1016/j.is.2016.03.008
0306-4379/© 2016 Elsevier Ltd. All rights reserved.

Although MapReduce and Spark frameworks have been praised for their high scalability and fault tolerance, they show poor performance in the case of data skew. There are important cases where a high percentage of processing in the reduce side ends up being done by only one node. Let us illustrate this by an example.

Example 1. Top accessed pages in Wikipedia. Suppose we want to analyze the statistics[1] that the free encyclopedia, Wikipedia, has published about the visits of its pages by users. In the statistics, for every hour, there is a file in which for each visited page, there is a line containing some information including, among others, its URL, language and the number of visits. Given a file, we want to return for each language, the top-k% accessed pages, e.g., top 1%.

To answer this query, we can write a simple program as in the following Algorithm[2]:

Algorithm 1. Map and reduce functions for Example 1.

```
map( $id : \mathcal{K}_1$, $content : \mathcal{V}_1$ )
    foreach $line \langle lang, page\_id, num\_visits, \ldots \rangle$ in $content$ do
        emit ($lang$, $page\_info = \langle num\_visits, page\_id \rangle$)
    end

reduce( $lang : \mathcal{K}_2$, $pages\_info : \text{list}(\mathcal{V}_2)$ )
    Sort $pages\_info$ by $num\_visits$
    foreach $page\_info$ in top $k\%$ do
        emit ($lang$, $page\_id$)
    end
```

In this example, the load of reduce workers may be highly skewed. In particular, the worker that is responsible for reducing the English language will receive a lot of values. According to the statistics published by Wikipedia,[3] the percentage of English pages over total was more than 70% in 2002 and more than 25% in 2007. This means for example that if we use the pages published up to 2007, when the number of reduce workers is more than 4, then we have no way for balancing the load because one of the nodes would receive more than 1/4 of the data. The situation is even worse when the number of reduce tasks is high, e.g., 100, in which case after some time, all reduce workers but one would finish their assigned task, and the job has to wait for the responsible of English pages to finish. In this case, the execution time of the reduce phase is at least equal to the execution time of this task, no matter the size of the cluster.

There have been some proposals to deal with the problem of reduce side data skew. One of the main approaches is to try to uniformly distribute the intermediate values to the reduce tasks, e.g., by dynamically repartitioning the keys to the reduce workers [3]. However, this approach is not efficient in many cases, e.g., when there is only one single intermediate key, or when most of the values correspond to one of the keys.

One solution for decreasing the reduce side skew is to filter the intermediate data as much as possible in the map side, e.g., by using a *combiner function*. However, the input of the combiner function is restricted to the data of one map task, thus its filtering power is very limited for some applications. Let us illustrate this by using our problem of top-1%. Suppose we have 1 TB of Wikipedia data, and 200 nodes for processing them. To be able to filter some intermediate data by the combiner function, we should have more than 1% of the total values of at least one key (language) in the map task. Thus, if we use the default splits of Hadoop (64 MB size), the combiner function can filter no data. The solution is to increase significantly the size of input splits, e.g. more than 10 GB (1% of total). However, using big splits is not advised since it decreases significantly the MapReduce performance due to the following disadvantages: (1) *more map-side skew*: with big splits, there may be some map tasks that take too much time (e.g. because of their slow CPU), and this would increase significantly the total MapReduce execution time; (2) *less parallelism*: big split size means small number of map tasks, so several nodes (or at least some of their computing slots) may have nothing to do in the map phase. In our example, with 10 GB splits, there will be only 100 map tasks, thus half of the nodes are idle. This performance degradation is confirmed by our experimental results reported in Section 5.11.

In this paper, we propose *FP-Hadoop*, a Hadoop-based system that uses a novel approach for dealing with the data skew in reduce side. In FP-Hadoop, there is a new phase, called *intermediate reduce* (*IR*), whose objective is to make the reduce side of MapReduce more parallel. More specifically, the programmer replaces his reduce function by two functions: *intermediate reduce* (*IR*) and *final reduce* (*FR*) functions. Then, FP-Hadoop executes the job in three phases, each phase corresponding to one of the functions: map, intermediate reduce (IR) and final reduce (FR) phases. In the IR phase, even if all intermediate values belong to only one key (i.e., the extreme case of skew), the reducing work is done by using the computing power of all available workers. Briefly, the data reducing in the *IR phase* has the following distinguishing features:

- *Parallel reducing of each key*: The intermediate *values of each key* can be processed in parallel by using multiple intermediate reduce workers.
- *Distributed intermediate block construction*: The input of each intermediate worker is a block composed of intermediate values *distributed over multiple nodes* of the system, and chosen using a *scheduling strategy*, e.g. locality-aware.
- *Hierarchical execution*: The processing of intermediate values in the IR phase can be done in several levels (iterations). This permits to perform *hierarchical execution plans* for jobs such as top-k% queries, in order to decrease the size of the intermediate data more and more.
- *Non-overwhelming reducing*: The size of the intermediate blocks is bounded by configurable maximum

[1] http://dumps.wikimedia.org/other/pagecounts-raw/

[2] This program is just for illustration; actually, it is possible to write a more efficient code by leveraging on the sorting mechanisms of MapReduce.

[3] http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia