Title

Zhouwang Fu[1] and Zhengwei Qi[1]
[1]*Shanghai Jiao Tong University*

Abstract

Shuffle is the term used to descirbe the cross-network read and aggregation of partitioned ancestor data before invoking reduce operation. As DAG computing framworks keep evolving, calculation and scheduling of each task are well optimized. However shuffle cuts off the data processing pipeline, introduce significant latency to successors. To remove shuffle overhead, we present XXX, a plugin system to decouple shuffle from DAG computing framework. XXX captures shuffle data in the memory and uses heuristic-FIFO scheduling to balance data blocks to elimite the explict barrier. We implement XXX and change Spark to use XXX as external shuffle service and scheduler. We evaluate XXX performance both on simulation and 50-machine Amazon EC2 cluster. Results show that, by incorporating XXX in Spark, the shuffle overhead can be reduce XXX.

1 Introduction

2 Motivation

In this section, we first study the shuffle pattern (2.1). Then we show the observation of the opportunities to optimize shuffle in 2.2

2.1 Characteristic of Shuffle

In DAG computing model such as Hadoop [1] and Spark [4], shuffle is dedicated to achieve a all-to-all data tranfer between stages.

It's I/O intensive, not CPU intensive. Write to disk for fault tolerance. Use network to transfer data.

It loosely couple with application context. Only blocks of bytes.

2.2 Observation

0. DAG computing have multi-rounds taks. Better load balance. Fit data blocks into main memory. Example Hadoop MapReduce suggests 10-100 maps per-nodes and 0.95 to 1.75 multiplied by no.node * no.container per node.[2]

1. Disk is not necessary for fault tolerance. Memory is very fast, Network can be fast. Example SSD 480 MB/S max. For 10Gbps network, 3X faster than SSD.

2. It's not efficient to put shuffle read/write operation into a task. Task is a unit of computation. But shuffle doesn't involve CPU

3. Shuffle can be easly decoupled from task. Based on these observations, it's straightforward to come up with a optimization to overlap the I/O operation of shuffle by leveraging multi-rounds property of DAG computing. In order to achieve this optimization, we have to decouple shuffle and perform pre-fetch as soon as each output of ancestor task is available. But is this feasible? We try to answer this question in the following sections.

3 Achieve Shuffle Pre-fetch

3.1 Decouple shuffle from task

Instead of writing shuffle data into disk, the data can be intercepted in memory and directly writen to network. But where?

3.2 Pre-schedule with Application Context

Naively random map tasks to end hosts. It's bad, may skew. We have perform more balanced allocation to avoid hurting the performance of successor tasks. Show the simulation with open cloud trace.

In order to acheive better balanced allocation, we should combine with application context and DAG.

For one shuffle, we can use first few tasks output to predict reduce distribution such as [3].

But results vary due to the input data distribution and partition function. Show three pics (hash parition and two range partition)

For hash partition function, in most scenrios are enough to have first completed tasks.

For range partition function and customed partition function, the relation between one output and the whole distribution can be oppsite. To avoid complex modification, we keep the partition function as a black box and

use weighted reservior sampling to prob the distribution. Show the prediction result and accurance.

For each prediction result, a percentage array of total data composition is calculated.

Combined with DAG information, i.e. other co-existing shuffle dependencies.

present pseudo code. (not completed yet).

4 Design

5 Evaluation

6 Conclusion

References

[1] Apache hadoop. `http://hadoop.apache.org/`.

[2] Apache hadoop tutorial. `http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html`.

[3] D. Cheng, J. Rao, Y. Guo, and X. Zhou. Improving mapreduce performance in heterogeneous environments with adaptive task tuning. In *Proceedings of the 15th International Middleware Conference*, pages 97–108. ACM, 2014.

[4] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.