Implementing a Key-Value Store based MapReduce Framework

Hirotaka Ogawa
AIST, Japan

Hidemoto Nakada
AIST, Japan

Tomohiro Kudoh
AIST, Japan

Abstract

MapReduce has been very successful in implementing large-scale data-intensive applications. Because of its simple programming model, MapReduce has also begun being utilized as a programming tool for more general distributed and parallel applications. However, its applicability is often limited due to relatively inefficient runtime performance and hence insufficient support for flexible workflows. In particular, the performance problem is not negligible in iterative MapReduce applications. In order to resolve such situations, we have been developing a new MapReduce prototype system called "SSS", which is based on distributed key-value store (KVS). In this poster, we present the design and implementation of SSS and the tentative benchmark results.

1 Introduction

We have been developing a new MapReduce prototype system called "SSS", which is based on distributed key-value store (KVS). Our main objectives is to bridge the gap between MapReduce data model and input/output data model, and to realize efficient executions not only for single-step MapReduce workloads but also more flexible workloads, including iterative applications.

SSS completely substitutes distributed KVS for the distributed file systems such as Hadoop DFS. Furthermore, SSS utilizes distributed KVS for storing the intermediate KV data, as well as the inputs of map tasks and the outputs of reduce tasks. SSS offers several advantages over existing MapReduce implementations:

1. We can bridge the gap between MapReduce data model and storage systems and handle KV data more intuitively.

2. We can eliminate shuffle & sort phase which may occupy the larger part of MapReduce execution time. Once all map tasks have finished storing intermediate KV data to the distributed KVS, all intermediate KV data have already been grouped by their own keys.

3. Map and reduce tasks can be realized as almost equivalent operations on top of the distributed KVS. This makes the implementation itself simple and enables any combination of multiple maps and reduces in a single workload.

Because of space limitation, we only describe our runtime design and the result of wordcount benchmark briefly. The details are described in [1]. And, in the poster session, we will present the latest benchmark results and the profiling data for the underlying storage system.

2 SSS MapReduce Runtime

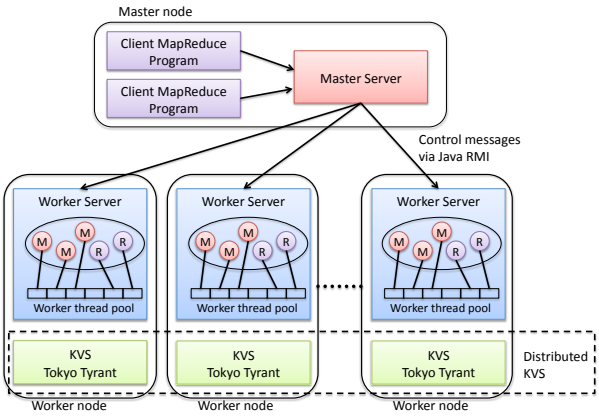

Figure 1: SSS MapReduce Runtime

As Figure 1 shows, SSS MapReduce runtime is built on top of the distributed KVS. Our distributed KVS provides a mechanism to partition KV data dynamically

across the set of storage nodes, based on consistent hashing. Each worker node serves both as a storage node of KVS and a worker server of MapReduce runtime. That is, each worker node is responsible for handling and executing map/reduce tasks for the KV data owned by itself. While all input/output data of map/reduce tasks are provided by distributed KVS, other commands and control data transfers to the worker servers are performed by SSS MapReduce runtime.

3 Wordcount Benchmark

We used an experimental cluster which consists of a master node and 16 worker nodes with 10Gbit Ethernet and ioDrive^{TM}Duo. Wordcount benchmark counts the number of occurrences of each word appeared in given multiple text files. We prepared 128 100MiB text files, therefore the total size of all files is 12.5GiB and each worker node has 800MiB text data.

Figure 2 shows the execution times of Hadoop, SSS, and packed-SSS (an optimized version of SSS). As you see, SSS is almost equivalent to or a little bit faster than Hadoop, and packed-SSS is almost 3.3 times faster than Hadoop.

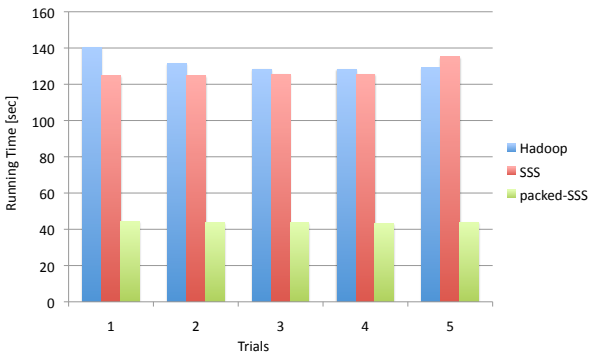

Figure 2: Running Times of Word Count

References

[1] OGAWA, H., NAKADA, H., TAKANO, R., AND KUDOH, T. SSS: An Implementation of Key-value Store based MapReduce Framework. In *Proceedings of 2nd IEEE International Conference on Cloud Computing Technology and Science (1st International Workshop on Theory and Practice of MapReduce)* (2010), pp. 754–761.