SSS : a MapReduce Framework based on Distributed Key-value Store

Hidemoto Nakada, Hirotaka Ogawa, Tomohiro Kudoh

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 HPC 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.

We implemented new MapReduce framework SSS based on distributed key-value store, that supports flexible workflows. Mappers and reducers read key-values only from its local storage enjoying high throughput and low latency.

We evaluated SSS comparing with Hadoop using synthetic benchmark and real application. The result showed that SSS is faster than Hadoop, except for simple sequential read from disks.

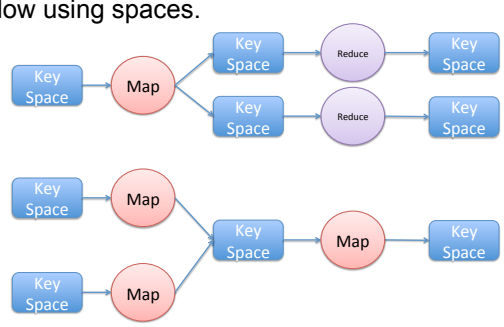
Key Ideas

Distributed KVS based

No distributed file system underneath, reducing abstraction layer that significantly slows down iterative jobs composed of hundreds of MR iterations. We use single node KVSs and tie them up with hashing.

Key Spaces for workflow

Keys are divided into several 'spaces'. Mappers and reducers are responsible for processing each one designated space. Programmers can construct arbitrary workflow using spaces.



Owner Computes Rule

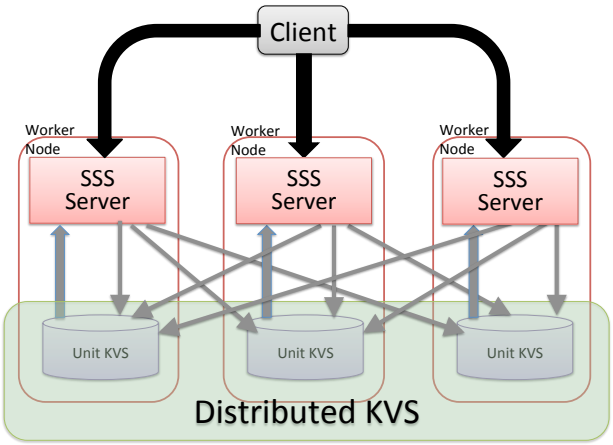
Key spaces spans all the nodes. Mappers and Reducers only process the data resides in the same node. It means that no inter-node communication happens before the computation. The output of Mappers/Reducers might be written to other nodes.

This simplifies scheduling a lot. In fact, SSS does not have so called scheduling module at all

Implementation

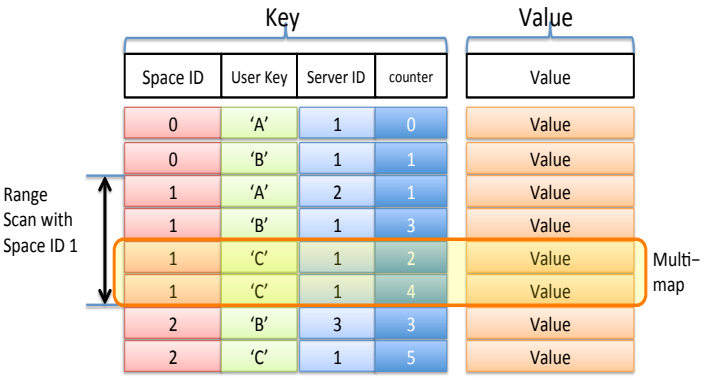
System Overview

The system is composed of one master node and worker nodes. Worker node hosts KVS as storage and processing server that runs Mappers and Reducers.



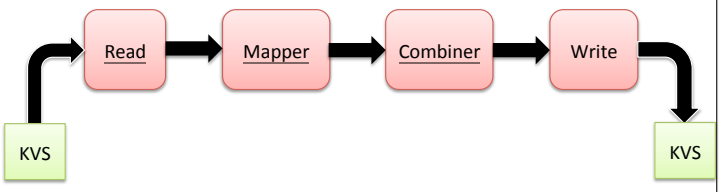
Key Spaces and Multi-Map implementation

Key-value Storage does not have 'space' or multi-map concept. We implemented space ID as key prefix so that we can use 'range scan' command to scan whole space. Multi-map is also handled with the key encoding. Server IDs and counters guarantee uniqueness of the Key, even though the User Key is the same.



Inside the Worker Server

Worker server is implemented in data-flow fashion, so that it can enjoy modern multi-core architecture and hide latencies to read and write data from/to KVS.



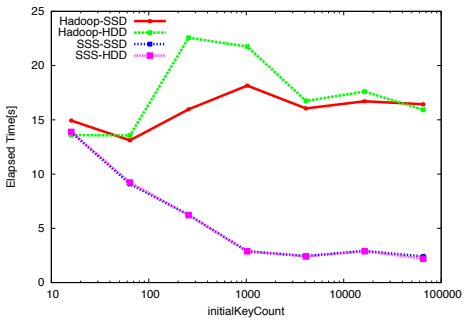
Evaluation with synthetic benchmark

To investigate the characteristics of SSS, we designed a synthetic benchmark suite that is composed of read, write, and shuffle intensive jobs. They input/output 16GB/1TB data in total during read, write and shuffle phase, respectively.

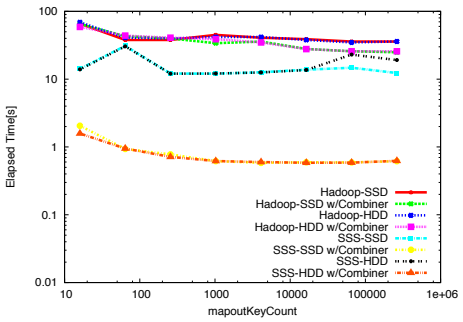
Evaluation Environment
- **Number of nodes: 16 + 1 (master)**
- **CPUs: Intel Xeon W5590 3.33GHz x 2**
- **Memory per node: 48GB**
- **Storage: Fusion-io ioDrive Duo 320GB**
- **NIC: Mellanox ConnectX-II 10G**

Total: 16G

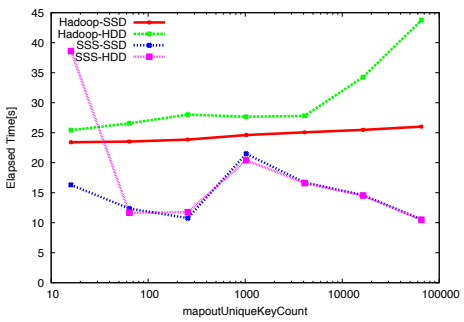
Read-Intensive



Shuffle-Intensive

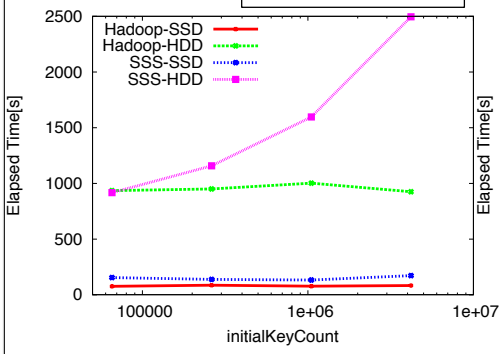


Write-Intensive

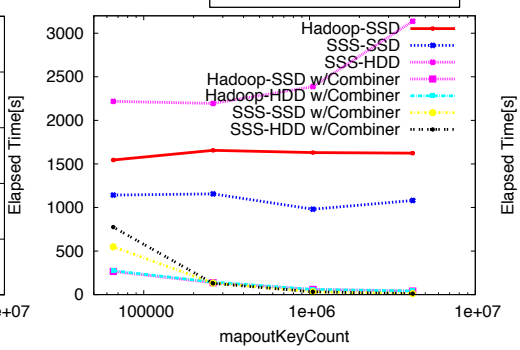


Total: 1T

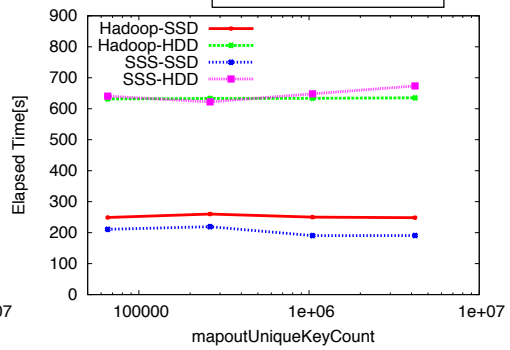
Read-Intensive



Shuffle-Intensive



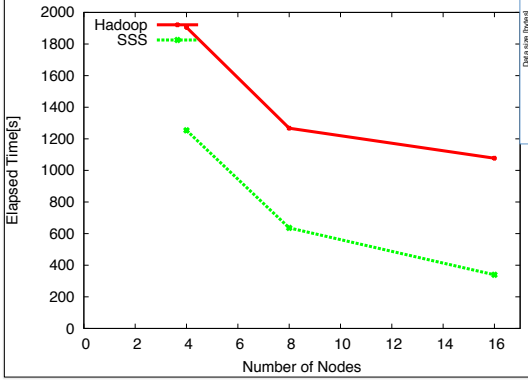
Write-Intensive

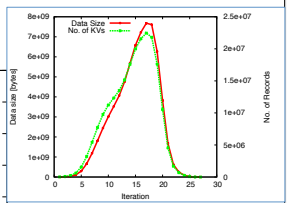


Evaluation with PrefixSpan

PrefixSpan is an algorithm for sequential pattern mining, which could be used, for example, to find frequently used idioms within a set of source code. MapReduce implementation of PrefixSpan repeats scanning through the whole dataset generating new dataset, until the dataset becomes empty.

- Input Data : 4M byte of source code.
- Nodes: 16
- Overhead for each iteration – job allocation.
  –Hadoop:11.3s, SSS: 0.4s , 300sec in Total.





Size of Dataset
for each iteration

Conclusion and Future work

The new MapReduce framework SSS is proposed and showed better performance for benchmark suite and a real application.

Our future work includes
- More Evaluation with various Applications, such as graph processing
- Supporting Continuous MapReduce
- Publish SSS as Open source software

Acknowledgement

We would like to express our deepest gratitude to Prof. Katsuro Inoue, Prof. Kenichi Hagihara, Prof. Masao Okita, and Mr. Yuki Inoue from Osaka University, for providing the PrefixSpan programs and the input data.

This work was partly funded by the New Energy and Indus- trial Technology Development Organization (NEDO) Green-IT project.

http://sss.apgrid.org