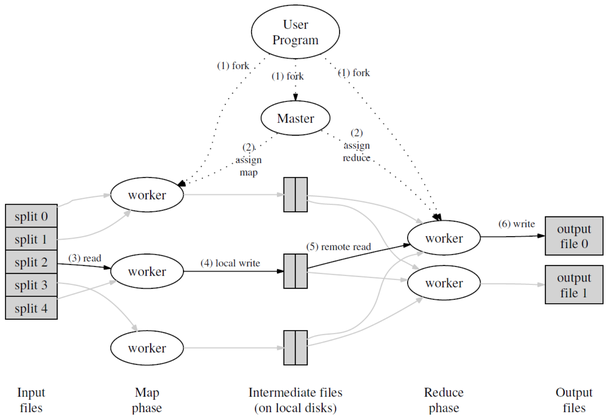
Lightweight MapReduce Framework

Zihao He, Shiming Zhuang
15618 final project

Summary

We implemented a lightweight MapReduce framework using C++ and demonstrated several machine learning algorithms (e.g. naïve bayes, logistic regression etc.) on top of it, which shows the feasibility of large-scale parallel machine learning algorithms.

Introduction



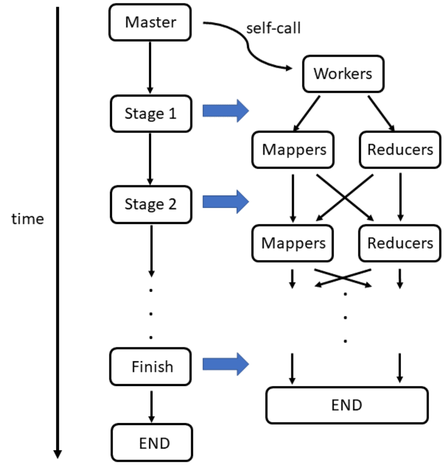
MapReduce is a framework where users specify a **map** function to generate a set of intermediate key/value pairs from input, and a **reduce** function to merge all intermediate values with the same key.

The framework will take care of the details such as partitioning the input data, scheduling and distributing tasks. This idea enables programmers without any experience with parallel systems to write **highly scalable** programs.

Many real-world problems can be expressed in this model, like word counting, reverse-weblink graph, machine learning algorithms.

Methodology

Our framework is based on distributed file system (e.g. AFS). The executing setup is just a cluster configuration file and a single binary executable.



The master is executed by the user and then it will ssh to other machines to bring up all the workers. The worker can turn into a mapper or reducer dynamically. After each stage there is a barrier for all the workers and only then the master will trigger the next stage.

The MapReduce Version of K-Means

Mappers:
Get the centroids from last iteration and read the input data. Calculate the Euclidean distance to each centroid and associate each instance with the closest centroid, and outputs (data instance id, cluster id).

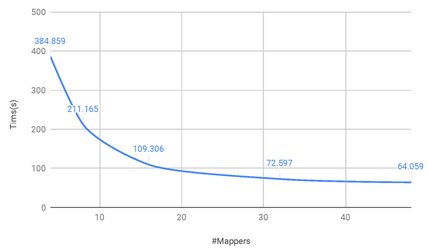
Reducers:
Calculate the average of the instances of each cluster, and output (cluster id, cluster centroid).

Results

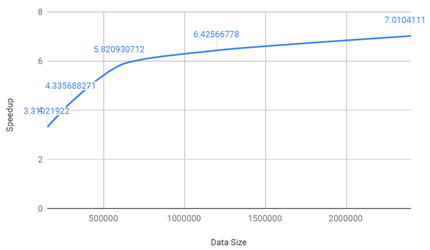
Sample code for K-Means:

```
#include "../src/mapreduce.h"
#include "../src/ml/kmeans.h"

using namespace lmr;
using namespace std;

int main(int argc, char **argv)
{
    MapReduceSpecification spec;
    MapReduceResult result;

    spec.config_file = "config.txt";
    spec.index = (argc == 2) ? atoi(argv[1]) : 0;
    spec.num_mappers = 10;
    spec.num_reducers = 10;

    MapReduce mr(&spec);
    ml::kmeans km(&mr);

    // 10 inputs, threshold is 0.1, maximum 20 iterations
    km.train("input_%d", 10, "centroids", 0.1, 20, result);
    km.predict("input_%d", 10, "result_%d", result);

    // print prediction time cost.
    printf("%.3fs elapsed.\n", result.timeelapsed);

    return 0;
}
```

Time vs. # of mappers:

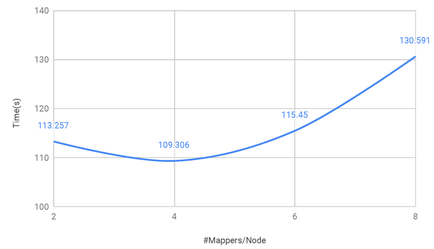


Speedup vs. # of inputs: (optimally 8x speedup)



Results

Time vs. # of mappers per node:



Conclusion

Features:

- **Scalability**: Easily implemented on multi-node system, and the scalability of the application is ideal when the problem size is large enough.
- **Dynamic Scheduling**: Supports dynamic scheduling for mappers and reducers. Can improve the work-load balance of the system.
- **Connection Reuse**: Connections between master and workers are reused in different rounds of MapReduce to avoid starting overhead.

Issues:

- **Starting Overhead**: For applications with few iterations, the overhead of workers assignment would overweigh the benefit of more workers.
- **Critical Section**: Master needs a mutex to protect the job queue, which must be done sequentially and may limit the scalability.

Reference

[1] J. Dean and S. Ghemawat, MapReduce: Simplied Data Processing on Large Clusters. Communications of the ACM, 2008.
[2] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng and K. Olukotun. Map-reduce for machine learning on multicore. Proceedings of the 19th International Conference on Neural Information Processing Systems, 2006.
[3] M. Bodoia. MapReduce Algorithms for k-means Clustering. CME 323: Distributed Algorithms and Optimization, Stanford. 2016.
[4] W. Cohen. Naive Bayes and Map-Reduce. 2017.