Tsinghua Science and Technology

Volume 18 | Issue 1 Article 1

2013

M2M: A Simple Matlab-to-MapReduce Translator for Cloud Computing

Junbo Zhang
School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China & the Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA

Dong Xiang
School of Software, Tsinghua University, Beijing 100084, China

Tianrui Li
School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China

Yi Pan
Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 01/12 pp1-9
Volume 18, Number 1, February 2013

M2M: A Simple Matlab-to-MapReduce Translator for Cloud Computing

Junbo Zhang, Dong Xiang, Tianrui Li, and Yi Pan*

Abstract: MapReduce is a very popular parallel programming model for cloud computing platforms, and has become an effective method for processing massive data by using a cluster of computers. X-to-MapReduce (X is a program language) translator is a possible solution to help traditional programmers easily deploy an application to cloud systems through translating sequential codes to MapReduce codes. Recently, some SQL-to-MapReduce translators emerge to translate SQL-like queries to MapReduce codes and have good performance in cloud systems. However, SQL-to-MapReduce translators mainly focus on SQL-like queries, but not on numerical computation. Matlab is a high-level language and interactive environment for numerical computation, visualization, and programming, which is very popular in engineering. We propose and develop a simple Matlab-to-MapReduce translator for cloud computing, called M2M, for basic numerical computations. M2M can translate a Matlab code with up to 100 commands to MapReduce code in few seconds, which may cost a proficient Hadoop MapReduce programmer some days on coding so many commands. In addition, M2M can also recognize the dependency between complex commands, which is always confusing during hand coding. We implemented M2M with evaluation for Matlab commands on a cluster. Several common commands are used in our experiments. The results show that M2M is comparable in performance with hand-coded programs.

Key words: MapReduce; Matlab; translator; cloud computing

1 Introduction

Nowadays, the volume of data is growing at an unprecedented rate. For the purpose of processing very large amounts of data, Google developed a software framework called MapReduce[1], to support large distributed applications using a large number of computers, collectively referred to as a cluster. As one of the most important cloud computing techniques, MapReduce has been a popular computing model for cloud computing platforms and is effective to analyze large amounts of data. Followed by Google's work, many MapReduce runtime systems have rapidly emerged. For example, (1) Apache Hadoop (Hadoop for short in following content)[2] was developed for data-intensive distributed applications. As an open source software framework, it helps construct the reliable, scalable, distributed systems. (2) Phoenix[3] is a shared-memory implementation of the MapReduce model for data-intensive processing tasks, which can be used to program multi-core chips as well as shared-memory multiprocessors (SMPs

• Junbo Zhang is with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China & the Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA. E-mail: JunboZhang86@163.com; jbzhang@cs.gsu.edu
• Dong Xiang is with the School of Software, Tsinghua University, Beijing 100084, China. Email: dxiang@tsinghua.edu.cn
• Tianrui Li is with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China. E-mail: trli@swjtu.edu.cn
• Yi Pan is with the Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA. E-mail: pan@cs.gsu.edu.
∗ To whom correspondence should be addressed.
Manuscript received: 2012-10-29; accepted: 2012-12-20

and ccNUMAs). (3) Aiming at providing a generic framework for developers to implement data- and computation-intensive tasks correctly, efficiently, and easily on the graphic processors (GPUs), Mars[4] is developed for GPUs using the MapReduce framework. Mars hides the programming complexity of the GPU behind the simple and familiar MapReduce interface. Hence, the developers can write their code on the GPU without any knowledge of the graphics APIs or the GPU architecture. (4) Iterative MapReduce model was introduced in Twister[5], which is a lightweight MapReduce runtime. It provides the feature for cacheable MapReduce tasks, allowing developers to develop iterative applications without spending much time on reading and writing large amount of data in each iteration. A version for Windows Azure called Twister4Azure[6] has also been released.

The MapReduce programming model used in current cloud computing has many limitations. The developers have to learn how to program with the MapReduce model and spend time on understanding characteristics of different cloud platforms. Without automatic parallelization software for many applications, the usage of cloud computing will be limited. In other words, we can not utilize massive computing power in the system unless we can perform effective distributed computing and program the applications easily. Automatic translation for certain programming languages is a possibility especially for many data-parallel application programming languages such as SQL and Matlab[7]. The MapReduce framework requires that users implement their applications by coding their own map and reduce functions. Although this low-level hand coding offers a high flexibility in programming applications, it increases the difficulty in program debugging and is also time consuming. High-level declarative languages can greatly simplify the effort on developing applications in MapReduce without hand-coding programs. Recently, several SQL-like declarative languages and their translators have been built and integrated with MapReduce to support these languages. For example, Hive[8] is a production SQL-to-MapReduce translator, which has greatly increased the productivity of writing database codes on cloud systems. Researchers also noticed that in practice the auto-generated MapReduce programs for many queries are often extremely inefficient compared to hand-optimized programs due to inefficient SQL-to-MapReduce translations which would create many unnecessary jobs. Hence, YSmart[9] is developed. It is a correlation aware SQL-to-MapReduce translator to translate SQL codes to Hadoop MapReduce codes and has been patched in Hive data warehousing system. YSmart significantly reduces redundant computations and I/O operations in the MapReduce execution.

However, SQL-to-MapReduce translators mainly focus on SQL-like queries, but not on numerical computation. Matlab[10] is a high-level language and interactive environment for numerical computation, visualization, and programming, which is very popular in engineering. We propose and develop a simple Matlab-to-MapReduce translator, called M2M, for basic numerical computation.

2 Background

This section reviews some basic ideas of MapReduce and relative features of Hadoop. MapReduce is a popular parallel model first introduced by Google, which is designed to handle and generate large scale data sets in distributed environment[1]. It provides a convenient way to parallelize data analysis process. Its advantages include conveniences, robustness, and scalability. The basic idea of MapReduce is to split the large input data set into many small pieces and assigned small task to different devices.

The process of MapReduce includes two major parts, Map function and Reduce function. The input files will be automatically split and copied to different computing nodes. After that the inputs will be sent to Map function in key-value pair format. The Map function will process the input pairs and also generate intermediate key-value pairs as inputs for Reduce function. The Reduce function will combine the inputs who have the same key and generate the final result. The final result will be written into the distributed file system. The users only need to implement Map and Reduce functions. They do not need to concern about how to partition the input which is automatically done by the model. The model will make tasks assigned evenly to every machine. Sometimes, to reduce communication overhead, a optional part Combine function can be employed. The Combine function can be regarded as a local Reduce, combines the results from Map function together locally, and returns a single value for Reduce function.

Hadoop[2] is an open source software framework which implements MapReduce model. It provides

a Hadoop Distributed File System (HDFS) as the global file system running on a cluster. Multiple independent MapReduce jobs can be executed at same time in Hadoop platform. These jobs can be scheduled using either fair sharing or First-In-First-Out (FIFO) scheduling. Users not only set up Hadoop on local clusters easily, but also on public cloud systems, such as Amazon EC2 and Microsoft Azure both support Hadoop through Amazon Elastic MapReduce[11] and HadoopOnAzure[12], respectively. The applications in this paper are implemented with Hadoop framework, and we use our local cluster as Hadoop platform.

3 Matlab-to-MapReduce

In this section, we give a simple method to translate Matlab code to MapReduce code and introduce our method from the following three aspects: (1) translate single Matlab command to MapReduce code; (2) translate multiple Matlab independent commands to MapReduce code; and (3) translate multiple Matlab dependent commands to MapReduce code.

Before this, we briefly introduce the flow chart of M2M, as shown in Fig. 1. It consists of the following three stages:

(1) The first stage is a lexical analyzer, used to the token generation. The scanner begins the analysis of the Matlab source code by reading the input, character by character, and grouping characters into meaningful words and symbols (tokens) defined by a grammar of regular expressions.

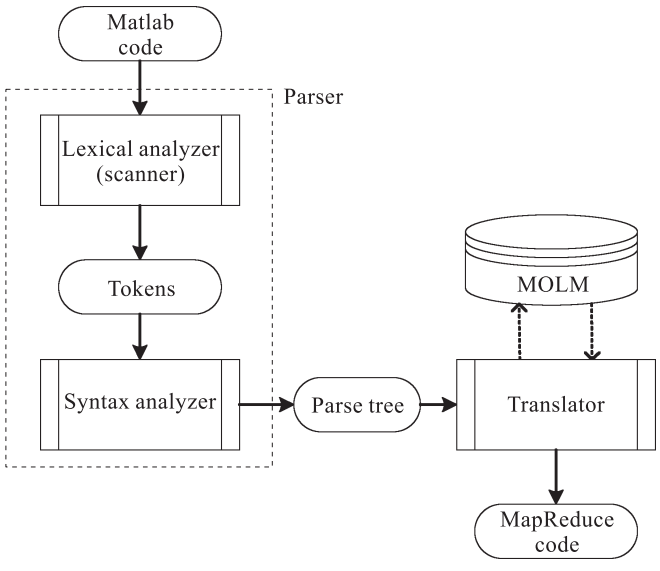


Fig. 1 Flow chart of M2M.

(2) The second stage is a syntax analyzer, used for parsing. Given a formal syntax specification (typically as a Context-Free Grammar (CFG)), the syntax analyzer reads tokens and groups them into units as specified by the productions of the CFG being used. Here, top-down parsing is adopted in our method.

(3) The third stage is a sematic analyzer (translator), used for semantic parsing or analysis. It is working out the implications of the expression just validated and taking the appropriate action. Here, aiming at the Matlab operations, a simple Math Operation Library based on MapReduce (MOLM) is built. When the translator processes a math command, it connects to the MOLM, gets the corresponding MapReduce code, and combines with a main function. It is worth noticing that the same commands only need one MapReduce code.

3.1 Single command to MapReduce

A single Matlab command is easily translated to MapReduce code. For example, there exists a Matlab command **min**, which returns the smallest elements along different dimensions of an array shown in Fig. 2. The M2M extracts the command **min** through the parser in Fig. 1. The translator in Fig. 1 assesses to the MOLM and gets the corresponding Map code, Combine code, and Reduce code according to the key word "min", which help generate Java class in Hadoop: Map class, Combine class, and Reduce class, respectively. Then, M2M helps generate the Job Configuration of single job for single command **min** in the **run** function. At last, by combining all generated codes with necessary Hadoop packages and imports, the corresponding Hadoop MapReduce code is generated, as shown in Fig. 3. Here, the input file is 'in.data', which should be uploaded to HDFS before running the application on Hadoop.

3.2 Multiple independent commands to MapReduce

A real application always contains multiple commands. In Fig. 4, given some multiple independent commands, M2M only translates to MapReduce codes and generates the Hadoop jobs one by one, and do not care about the relations between jobs. MapReduce

Matlab code using the command **min**.

```
x = load('in.data') //load data from the file 'in.data'
a = min(x)
```

Fig. 2 Matlab code using the command min.

MapReduce code for the command **min** (Renamed the application as **AutoMin**)

```
package ...
import ...
public class AutoMin extends Configured implements Tool {
    public static class MinMap extends Mapper<Object, Text, Text, DoubleWritable> {
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            Integer cc = 1;
            while (itr.hasMoreTokens()) {
                context.write(new Text(cc.toString()), new DoubleWritable(new Double(itr.nextToken())));
                ++cc;
            }
        }
    }
    public static class MinCombine extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
        private DoubleWritable result = new DoubleWritable();
        public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
                        throws IOException, InterruptedException {
            Double mmin = Double.MAX_VALUE;
            for (DoubleWritable val : values) {
                Double tmp = val.get();
                if (mmin > tmp) mmin = tmp;
            }
            result.set(mmin);
            context.write(key, result);
        }
    }
    public static class MinReduce extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
        private DoubleWritable result = new DoubleWritable();
        public void reduce(Text key, Iterable<DoubleWritable> values,
                Context context) throws IOException, InterruptedException {
            Double mmin = Double.MAX_VALUE;
            for (DoubleWritable val : values) {
                Double tmp = val.get();
                if (mmin > tmp) mmin = tmp;
            }
            result.set(mmin);
            context.write(key, result);
        }
    }
    public int run(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        Job job = new Job(conf, "AutoMin");
        job.setJarByClass(AutoMin.class);
        job.setMapperClass(MinMap.class);
        job.setCombinerClass(MinCombine.class);
        job.setReducerClass(MinReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1] + "#1"));
        ControlledJob cj1 = new ControlledJob(job.getConfiguration());
        JobControl JobCtrl = new JobControl("JobGroup");
        JobCtrl.addJob(cj1);
        Thread thread = new Thread(JobCtrl);
        thread.start();
        while (!JobCtrl.allFinished()) {
            try { Thread.sleep(1000);
            } catch (InterruptedException e) { e.printStackTrace();
            }
        }
        JobCtrl.stop();
        return 1;
    }
    public static void main(String[] args) throws Exception {
        int ret = ToolRunner.run(new Configuration(), new AutoMin(), args);
        System.exit(ret);
    }
}
```

Fig. 3 MapReduce code for the command min.

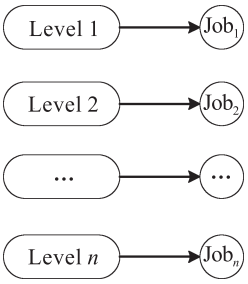


Fig. 4 Independent commands to independent jobs.

can help give efficient data parallelism applications. Here, since jobs are independent with each other, the characteristic of task parallelism is added to generate Hadoop MapReduce codes, which can help accelerate the program.

Figure 5 gives a Matlab pseudo-code with multiple independent commands, where **min**, **max**, and **sum** are Matlab commands. The M2M detects these commands in sequence and translates these command one by one. For any one command, the process of translation is similar to that in Section 3.1. The difference is that these jobs are combined as a group, which is the key step for task parallelism. The corresponding Hadoop Job Configuration is shown in Fig. 6. When executing a Hadoop application, the **main** function in Hadoop will call the **run** function.

3.3 Multiple dependent commands to MapReduce

For more complex commands, there exist the dependencies among them. For example, the Matlab command **std** is used to compute the standard deviation.

```
Matlab pseudo-code
x = load('in1.data') //load data from the file 'in1.data'
y = load('in2.data') //load data from the file 'in2.data'
o1 = min(x)
o2 = max(x)
o3 = sum(x)
o4 = min(y)
```

Fig. 5 Matlab pseudo-code with multiple independent commands.

```
Hadoop Job Configuration (in run function)
//Job-o1 is for o1 in Fig. 5
ControlledJob Job-o1, Job-o2, Job-o3, Job-o4;
JobControl JG; // JG is a group of jobs
JG.addJobCollection(Job-o1, Job-o2, Job-o3, Job-o4);
JG.run();
```

Fig. 6 Hadoop Job Configuration with multiple independent jobs.

The command **std** uses the sample standard deviation by default, which denoted by s and defined as follows:

$$s = \left(\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2 \right)^{\frac{1}{2}} \quad (1)$$

where $\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$ and n is the number of elements in the sample.

Hence, the mean of elements in the sample should be computed first. We set the sequence for jobs when running. Figure 7 shows this application by using a level view. Level 0 is to read the data; Level 1 is to compute the mean; Level 2 is to compute the standard deviation.

Next, we give a more detailed example through the level view. Figure 8 gives a Matlab pseudo-code with multiple dependent commands, where F11, F12, F13, F14, F21, F22, and F31 are Matlab commands. The M2M detects these commands in sequence, and generates dependency graph first, which can be shown as the multi-level view in Fig. 9. Here, we also use task parallelism to accelerate the program. The jobs in each level can be executed at the same time. But the jobs in level $k+1$ would be executed only after all jobs in level k are completed. In general, arrows represent data dependency. For example, Job_{31} depends on Job_{21}, Job_{22}, and Job_{13}. We here give the corresponding Job

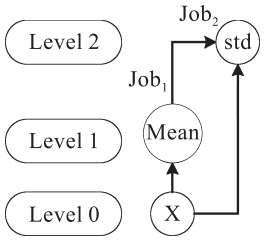


Fig. 7 The Matlab command std: 2-level.

```
Matlab pseudo-code
x = load('in1.data') //load data from the file 'in1.data'
y = load('in2.data') //load data from the file 'in2.data'
z = load('in3.data') //load data from the file 'in3.data'
o11 = F11(x)
o12 = F12(x,y)
o13 = F13(z)
o14 = F14(z)
o21 = F21(o11,o12)
o22 = F22(z,o14)
o31 = F31(o21,o13,o22)
```

Fig. 8 Matlab pseudo-code with multiple dependent commands.

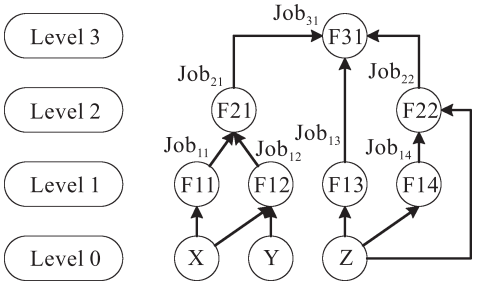


Fig. 9 Multi-level dependency graph.

Configuration in the **run** function of Hadoop, as shown in Fig. 10. When executing a Hadoop application, the **main** function in Hadoop will call the **run** function.

4 Experimental Analysis

Our experiments were carried out on a cluster using five nodes. Each node has 16 GB main memory and uses AMD Opteron(TM) Processor 2376 CPUs (8 cores, each has a clock frequency of 2.3 GHz). One node is a master node, which is used to run JobTracker. And, another four nodes are used to run TaskTracker. Each of that is configured to provide 8 task slots, including 4 map slots and 4 reduce slots. It means there are total 16 map slots and 16 reduce slots. Hadoop 1.0.1 and JDK 1.7.0_05 are used for the MapReduce runtime system. In the following experiments, the Mablab operations are all about arraies. Thus the matrix data can be used to run all these operations. Here, a $200\,000 \times 1000$ matrix is randomly generated as our test data set, and its size is 933 MB. Before running a job in Hadoop, we should upload the data set to HDFS. Since the default block size in HDFS is 64 MB, our test data is partitioned into

Hadoop Job Configuration (in **run** function)

```
ControlledJob J11, J12, J13, J14, J21, J22, J31;
J21.addDependingJob(J11);
J21.addDependingJob(J12);
J22.addDependingJob(J14);
J31.addDependingJob(J21);
J31.addDependingJob(J13);
J31.addDependingJob(J22);
JobControl JG; // JG is a group of jobs
JG.addJobCollection(J11, J12, J13, J14, J21, J22, J31);
JG.run();
```

Fig. 10 Hadoop Job Configuration with multiple dependent jobs.

$\left\lceil \frac{933}{64} \right\rceil = 15$ blocks in HDFS.

4.1 Job scheduling in Hadoop

Hadoop provides a pluggable MapReduce scheduler that provides a way to share large clusters. There are several common schedulers: Capacity Scheduler, Fair Scheduler, and FIFO Scheduler. The default scheduling mode in Hadoop is FIFO, which is used to schedule the jobs in our experiments. FIFO describes the principle of a queue processing technique or servicing conflicting demands through an ordering process by First-Come, First-Served (FCFS) behavior: where the Hadoop Jobs leave the queue in the order as they arrive, or wait one's accomplishment. In the experiments, there are 16 map slots in our Hadoop platform and 15 data blocks. For such 15 data blocks, initially, 15 Map tasks are created at default when a job (Matlab command) is submitted to the run-time. Since there are 16 cores (16 tasks) for Map tasks, one core is idle and can be allocated to the next job (Matlab command). Then FCFS allocation is implemented for the following commands. It is similar for Reduce tasks.

The Job scheduler used in the experiments is simple. It is not perfect for load balancing. The better scheduling algorithms have been introduced for different applications in literatures. For example, Longest Approximate Time to End (LATE) is designed to improve MapReduce performance in heterogeneous environments[13]. To predict and manage the performance of workloads in a shared environment, a task scheduler based on deadline scheduler is proposed, which can dynamically predict the performance of concurrent MapReduce jobs and adjusts the resource allocation for the jobs[14]. Deadline constraint scheduler for Hadoop ensures that only jobs whose deadlines can be met are scheduled for execution[15].

4.2 A comparison of M2M and hand-coded program

We compare M2M and hand-coded programs on single Matlab command. Here, we just use several common commands in our experiments (see Table 1). The generated matrix is used as input A.

Figure 11 shows the results of M2M and hand-coded programs. Each bar shows the execution time of the jobs. We found the hand-coded program and M2M almost cost the same time. M2M is only about 5%

Table 1 Matlab commands.

Command	Syntax	Description
min	R=min(A)	min(A) returns the smallest elements along different dimensions of an array. If A is a vector, min(A) returns the smallest element in A. If A is a matrix, min(A) treats the columns of A as vectors, returning a row vector containing the minimum element from each column.
max	R=max(A)	max(A) returns the largest elements along different dimensions of an array. If A is a vector, max(A) returns the largest element in A. If A is a matrix, max(A) treats the columns of A as vectors, returning a row vector containing the maximum element from each column.
mean	R=mean(A)	mean(A) returns the mean values of the elements along different dimensions of an array. If A is a vector, mean(A) returns the mean value of A. If A is a matrix, mean(A) treats the columns of A as vectors, returning a row vector of mean values.
length	R=length(A)	length(A) finds the number of elements along the largest dimension of an array.
sum	R=sum(A)	sum(A) returns sums along different dimensions of an array. If A is a vector, sum(A) returns the sum of the elements. If A is a matrix, sum(A) treats the columns of A as vectors, returning a row vector of the sums of each column.
std	R=std(A)	If A is a vector, std(A) returns the standard deviation. If A is a matrix, std(A) returns a row vector containing the standard deviation of the elements of each column.

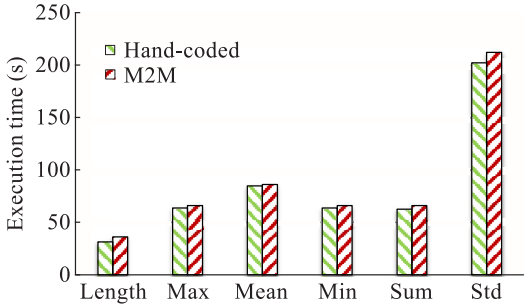


Fig. 11 A comparison of M2M and hand-coded programs on single command.

slower, which is caused by the JobControl of Hadoop. Hand-coded program does not always need JobControl. Instead, M2M uses JobControl to control these jobs. This difference makes M2M a little slower.

4.3 A comparison of M2M with- and without-task-parallelism program on multiple independent commands

M2M can generate a cloud program with-task-parallelism by using JobControl. But, one-command-to-one-job translation will not create task parallelism. We compare the M2M with- and without-task-parallelism programs on multiple independent commands, as shown in Fig. 12. Here, the commands in Table 1 are used repeatedly but without data dependency between commands. Obviously, with-task-parallelism program has a better performance, in especial to deal with more commands. Here, we have not compared M2M with hand-coded program because it would spend much time on coding so many commands with MapReduce.

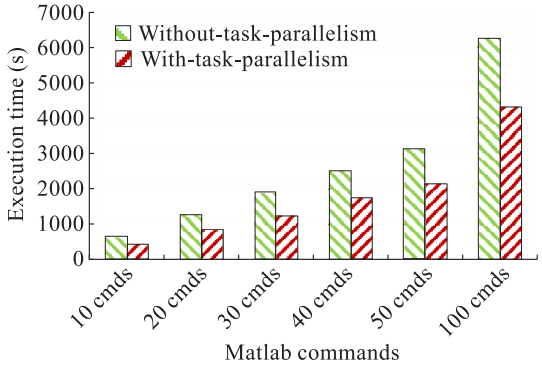


Fig. 12 A comparison of M2M with- and without-task-parallelism program on multiple independent commands.

4.4 A comparison of M2M with- and without-task-parallelism program on multiple dependent commands

We compare the M2M with- and without-task-parallelism program on multiple dependent commands as shown in Fig. 13. Here, the commands in Table 1 are used repeatedly with data dependency between commands. The advantage of M2M with-task-parallelism program becomes more obvious because the dependent jobs would output a smaller size of data, which would be the input of subsequent jobs. Program without-task-parallelism program only executes these jobs with small data one by one. On the contrary, M2M executes these jobs at the same time. It means that if there are enough computing resources, program without-task-parallelism would execute k times for k jobs but M2M would execute only one time.

5 Conclusions

In this paper, we proposed a simple method for translating Matlab code to MapReduce code and

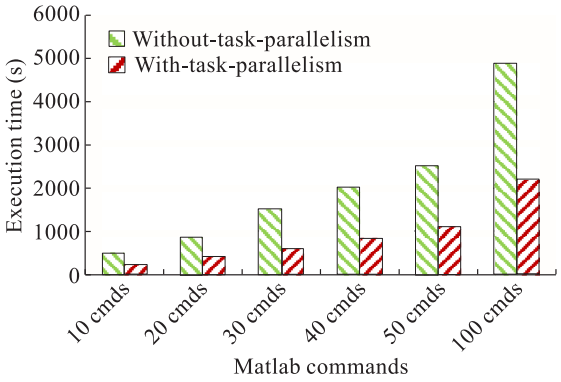


Fig. 13 A comparison of M2M with- and without-task-parallelism program on multiple dependent commands.

developed a simple Matlab-to-MapReduce translator called M2M. The experiments show that M2M is comparable with the performance of hand-coded programs. M2M provides not only data parallelism but also task parallelism. Moreover, M2M can help Matlab programmers easily deploy applications on cloud systems without learning programming with MapReduce, and help proficient Hadoop MapReduce programmers greatly reduce coding time. For all that, at present, M2M is still in early stage and only supports some basic Matlab commands. It does not support visualization and complex matrix operations, which both are challenges for massive data. One of our future task is to enrich our math operation libraries based on MapReduce. We plan to use the open source math operation libraries based on MapReduce, such as Mahout[16], which provides partial math operations on top of Apache Hadoop. M2M can not process loop commands by now, therefore, another future work is to enhance M2M and make it support loop commands. Now, the Job scheduler used in M2M is very simple. It is not perfect for load balancing. We will employ the better schedulers to improve M2M.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (Nos. 61175047, 61100117, and 61202043) and the US National Science Foundation (No. OCI-1156733).

References

[1] J. Dean and S. Ghemawat, Mapreduce: Simplified data processing on large clusters, *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, Jan. 2008.

[2] T. White, *Hadoop: The Definitive Guide*, 2nd ed. O'Reilly Media / Yahoo Press, 2010.

[3] J. Talbot, R. M. Yoo, and C. Kozyrakis, Phoenix++: Modular mapreduce for shared-memory systems, in *Proc. of the Second International Workshop on MapReduce and Its Applications*, New York, NY, USA: ACM, 2011, pp. 9-16.

[4] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, Mars: A mapreduce framework on graphics processors, in *Proc. of the 17th International Conference on Parallel Architectures and Compilation Techniques*, New York, NY, USA: ACM, 2008, pp. 260-269.

[5] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, Twister: A runtime for iterative mapreduce, in *Proc. of the 19th ACM Int. Symposium on High Performance Distributed Computing*, New York, NY, USA: ACM, 2010, pp. 810-818.

[6] T. Gunarathne, B. Zhang, T.-L. Wu, and J. Qiu, Portable parallel programming on cloud and hpc: Scientific applications of twister4azure, in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE Int. Conf. on*, Dec. 2011, pp. 97-104.

[7] Y. Pan and J. Zhang, Parallel programming on cloud computing platforms: Challenges and solutions, *KITCS/FTRA Journal of Convergence*, vol. 3, no. 4, pp. 23-28, Dec. 2012.

[8] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, Hive: A warehousing solution over a map-reduce framework, *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1626-1629, Aug. 2009.

[9] R. Lee, T. Luo, Y. Huai, F. Wang, Y. He, and X. Zhang, Ysmart: Yet another sql-to-mapreduce translator, in *Distributed Computing Systems (ICDCS), 2011 31st Int. Conf. on*, June 2011, pp. 25-36.

[10] A. Gilat, *MATLAB: An Introduction with Applications*, 4th ed. John Wiley & Sons, 2011.

[11] Amazon Elastic MapReduce, http://aws.amazon.com/elasticmapreduce/, 2012.

[12] HadoopOnAzure, https://www.hadooponazure.com/, 2012.

[13] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, Improving mapreduce performance in heterogeneous environments, in *Proc. of the 8th USENIX Conf. on Operating Systems Design and Implementation*, Berkeley, CA, USA: USENIX Association, 2008, pp. 29-42.

[14] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguadé, M. Steinder, and I. Whalley, Performance-driven task co-scheduling for mapreduce environments, in *Network Operations and Management Symposium (NOMS), 2010 IEEE*, April 2010, pp. 373-380.

[15] K. Kc and K. Anyanwu, Scheduling hadoop jobs to meet deadlines, in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second Int. Conf. on*, Dec. 2010, pp. 388-392.

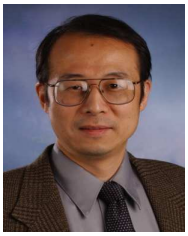
[16] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in Action*. Greenwich, CT, USA: Manning Publications Co., 2011.

Junbo Zhang received his B.Eng. degree from Southwest Jiaotong University, China, in 2009. He is currently a PhD candidate at the School of Information Science and Technology, Southwest Jiaotong University, China. He is also a visiting PhD student at the Department of Computer Science, Georgia State University, USA. He is interested in data mining, cloud computing, granular computing, and rough sets. He is now a student member of ACM and CCF. Since 2009, he has published over 20 research papers in referred journals (e.g., Information Sciences, International Journal of Approximate Reasoning, International Journal of Intelligent Systems) and conferences.

Dong Xiang received his BS and MS degrees in Computer Science from Chongqing University in 1987 and 1990, respectively. He got his PhD degree in Computer Engineering from the Institute of Computing Technology, the Chinese Academy of Sciences in 1993. He visited Concordia University, Canada as a Postdoctor from 1994 to 1995, and the Coordinated Science Lab., University of Illinois, Urbana Champaign from 1995 to 1996. He was with Institute of Microelectronics, Tsinghua University as an Associate Professor from 1996.10 to 2003.3. He moved to School of Software, Tsinghua University Mar. 2003. He is now a Professor of the School of Software, Tsinghua University. He was with Nara Institute of Science and Technology, Nara, Japan as a JSPS invitation fellow from Apr. 2003 to Sept. 2003. He is a Senior Member of IEEE.

Tianrui Li received his BS, MS, and PhD degrees from the Southwest Jiaotong University, China in 1992, 1995, and 2002, respectively. He was a Post-Doctoral Researcher at Belgian Nuclear Research Centre (SCK • CEN), Belgium from 2005-2006, a visiting professor at Hasselt University, Belgium in 2008 and the University of Technology, Sydney, Australia in 2009. And, he is presently a Professor and the Director of the Key Lab of Cloud Computing and Intelligent Technique of Sichuan Province, Southwest Jiaotong University, China.

Since 2000, he has co-edited 4 textbooks, 10 proceedings, 5 special issues of international journals and published over 100 research papers in refereed journals and conferences. He is a senior member of IEEE, the vice chair of IEEE CIS Chengdu Chapter and the area editor of International Journal of Computational Intelligence Systems (IJCIS, SCI indexed). He has served as ISKE2007, ISKE2008, ISKE2009, ISKE2010, ISKE2011, JRS2012 program chairs, IEEE GrC 2009 program vice chair and RSKT2008, FLINS2010 organizing chairs and has been a reviewer for several leading academic journals.



Yi Pan is the chair and a professor in the Department of Computer Science at Georgia State University. He is also a "Changjiang Scholar" Chair Professor at Central South University, an IV-endowed visiting chair professor at Tsinghua University, and a guest professor at Beijing University in China. Dr. Pan received his BEng and MEng degrees in computer engineering from Tsinghua University, China, in 1982 and 1984, respectively, and his PhD degree in computer science from the University of Pittsburgh, USA, in 1991. His profile has been featured as a distinguished alumnus in both Tsinghua Alumni Newsletter and University of Pittsburgh CS Alumni Newsletter. Dr. Pan's research interests include parallel and cloud computing, wireless networks, and bioinformatics. Dr. Pan has published more than 150 journal papers with over 50 papers published in various IEEE journals. In addition, he has published over 150 papers in refereed conferences. He has also co-authored/co-edited 37 books. His work has been cited more than 3600 times. Dr. Pan has served as an editor-in-chief or editorial board member for 15 journals including 6 IEEE Transactions. He is the recipient of many awards including IEEE Transactions Best Paper Award, IBM Faculty Award, JSPS Senior Fellowship, IEEE BIBE Outstanding Achievement Award, NSF Research Opportunity Award, and AFOSR Summer Faculty Research Fellowship. He has organized many international conferences and delivered more than 20 keynote speeches at various international conferences.