H2T: A simple Hadoop-to-Twister Translator for Cloud Computing

Junbo Zhang*†, Jian-Syuan Wong†, Yi Pan† and Tianrui Li*
*School of Information Science and Technology
Southwest Jiaotong University, Chengdu 610031, China
Email: JunboZhang86@163.com, trli@swjtu.edu.cn
†Department of Computer Science
Georgia State University, Atlanta, GA 30303, USA
Email: jwong9@student.gsu.edu, pan@cs.gsu.edu

***Abstract*—MapReduce has become one of the most popular programming model for big data analysis in cloud systems due to its simplicity for implementing data parallel applications. There are several platforms for users to develop their applications based on MapReduce framework such as Hadoop and Twister. Hadoop is one of the most popular runtime systems for MapReduce applications and supported by various organizations, however, the original design for Hadoop did not propose an iterative feature efficiently which is required for many scientific applications. Twister, another system for Iterative MapReduce, is introduced and designed to facilitate iterative applications based on MapReduce framework. It has shown that Twister has the better performance than Hadoop on some applications such as Pairwise Distance Calculation (Smith Waterman Gotoh distance). Automatic translations between two program languages in cloud platforms can help developers move their applications from one cloud to anther cloud without changing codes. In this paper, we propose a simple Hadoop-to-Twister translator named H2T which is designed for converting simple Hadoop applications into Twister applications. The experimental results show that translated Twister applications is much faster than original Hadoop applications.**

***Keywords*-MapReduce; translator; Hadoop; Twister; cloud computing.**

I. Introduction

Owing to the advancement of information technology, a large mount of data is generated every day in various areas such as information retrieval and astronomy. Large amount of time is required to handle this data. Therefore, there are the urgent needs to find a method to analyze the large data set efficiently. More and more computing resources are required as well.

Cloud computing is a novel technology for users to access computing resources and services through internet connections. It provides users the scalable computing resources that allow users to utilize as much computing power as they need. The current cloud computing systems provide three major services [1]:

1) Infrastructure as a service (IaaS): Hardware resources are provided through the cloud services that allow users to use a real machine or virtual machine based on the service.
2) Platform as a service (PaaS): Add a layer to the infrastructure, providing a platform usually includes the operating system and program language execution environment where applications can be written and deployed.
3) Software as a service (SaaS): Refers to applications running on cloud infrastructures, typically services are provided to the end user through a web browser.

MapReduce is a data parallel programming model introduced by Google [2] which has several attractive features allow developers to analyze large data efficiently. Computation in MapReduce model includes two major phase: Map phase and Reduce phase. Each phase utilizes key/value pairs as the input and output. The programmer should choose the data type of key and value based on the different requirements. Additionally, the map and reduce function is also required for Map and Reduce phase. At first, input data is taken by master node and divided into sub-program for map phase. The map function can take the partitioned input data in the first set of key/value pairs ($K1, V1$) format, and also generate the output in another set of key/value pairs ($K2, V2$). The intermediate results then are collected by the runtime system and produce the key/list values ($K2$, a list of ($V2$)) pairs as input of reduce function. Once the reduce function finishes its process, the key/list pairs ($K3$, a list of ($V3$)) are created as the output.

After Google's work for the MapReduce programming model, various MapReduce runtime systems are proposed for users to develop their applications based on the data parallel programming model, such as Phoenix [3], Apache Hadoop [4], Mars [5] and Twister [6].

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 [7]. In other words, we cannot utilize massive computing power in the system unless we can perform efficient distributed computing and program the applications easily. Automatic translation for certain programming languages is

978-0-7695-5010-7/13 $26.00 © 2013 IEEE
DOI 10.1109/ISBAST.2013.32

a possibility especially for many data-parallel application programming languages such as SQL and Matlab. X-to-MapReduce (X is a programming language) can help developers move local applications to the cloud such as Hive [8], YSmart [9] and M2M [10]. As well, automatic translations between two program languages in cloud platforms can help developers move their applications from one cloud to anther cloud without changing codes.

Even though the most applications built on Twister have much better performances than on Hadoop, many users still choose Hadoop as the platform for their implementations. The main reason why Hadoop has become so popular is that it not only works on private computer clusters, but also is supported by many popular cloud computing services such as Amazon Elastic MapReduce (Amazon EMR) [11] and Microsoft Windows Azure [12]. For another reason, we can assume Twister is a more complicated platform than Hadoop for implementing applications since it requires much more work for programming than Hadoop does.

In Table I, there are five applications which are the popular ones such as "**Word Count**", "**Page Rank**", "**K-Means Clustering**". The source codes of these Hadoop applications are provided by Hadoop-1.0.1 (http://hadoop.apache.org/) and HaLoop [13], and the Twister codes are provided by twister-0.9 (http://www.iterativemapreduce.org/). The rest two applications, **Maximum** and **Sum**, are implemented by ourselves for calculating the maximum and sum value at each row of an array. It is obvious to tell that Twister applications has a larger size of source codes than Hadoop applications which indicates that users need to write more codes for the same applications with Twister. A simple Hadoop-to-Twister translator (H2T) is proposed in this paper to help Hadoopers (Hadoop programmers) run their applications on both platforms.

Table I
CODE SIZE COMPARISON IN LINES OF SOURCE CODE FOR WORKLOADS IN HADOOP (H) AND TWISTER (T)

	Map		Reduce		Other		Total	
	H	T	H	T	H	T	H	T
WordCount	13	53	14	18	33	106	60	177
Maximum	12	33	15	19	38	101	65	153
Sum	10	32	11	17	36	102	57	151
PageRank	98	113	84	59	114	250	296	422
K-Means	87	60	35	49	126	150	248	259

The rest of this paper is organized as follows. Section II introduces two MapReduce runtime systems: Hadoop and Twister. Section III gives a simple method for translating Hadoop codes to Twister codes. Section IV presents an example for the WordCount application on these two systems. Section V show the experimental analysis based on the performance of hand-coded Hadoop and Twister codes as well as the translated Twister codes. The paper ends with conclusions and future work in Section VI.

II. BACKGROUND

A. Hadoop

Apache Hadoop [4] is an open-source software framework designed for data processing across massive computing clusters and nodes which also supports MapReduce model programming. Hadoop Distributed File System (HDFS) is available on the framework to create a global file systems running on a cluster. Input data is firstly divided into several data chunks as the input data on HDFS for map task. The TaskTracker on the node then retrieves key/value pairs from the data chunk and assigns them to Mappers. The output key/value pairs of the map function are sorted and stored locally, and passed to Reducers as the input data through HTTP. Once the process is finished in reduce function, the final result is delivered to HDFS through network.

B. Twister

Twister [6] is an iterative MapReduce framework, mainly designed for facilitating iterative applications based on MapReduce programming model. Twister demonstrates that it is a good fit for several scientific applications such as Blast (an application comparing primary biological sequence information). To the support of iterative applications, Twister also provides other attractive features that are different to typical MapReduce runtime systems:

1) Intermediated results generated by Mappers and Reducers are stored in the distributed memory of local workers which are more efficient than writing and reading intermediate data in local disk repeatly.
2) Input file is partitioned into several smaller files as the input data of Mappers which offers users more flexibility to manipulate the input data.
3) Apply publish/subscribe messaging infrastructure as the network broker to make communications become more efficient.
4) Combiner: the combine phase in Twister has different functionalities with the one in Hadoop; the main purpose for combiner in Twister is used to collect results from all the Reduce outputs and generate the only final result in combine phase. See Figure 1.

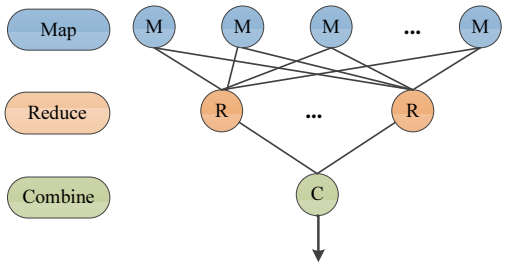

Figure 1. The flowchart of MapReduce in Twister

Before starting the MapReduce job in Twister, users are asked to split the input file into portioned files manually. Partitioned files then are taken by the Mappers as input data. Therefore, to average the workload for each processor, we usually choose the number of split files as multiples of the number of total processors.

III. Hadoop-to-Twister

In the section, we present the method for converting Hadoop codes into Twister codes. Hadoop and Twister are both implementations of MapReduce using the same programming language JAVA. Nonetheless, there still exist differences between these two runtime systems.

At the beginning of our work, several code templates are created for the different requirements of Twister framework which include the templates for "Import Library", "Map Class", "Reduce Class", and "Main Function". With these templates, the translator now only focus on converting developers' Hadoop logics into Twister without concerning about the differences between these two MapReduce frameworks. Take Map class of Hadoop and Twister as an example, the input data of Hadoop is firstly divided into several blocks automatically, and each default block size is 64MB. The map function of Hadoop then handles a single line of input data at a time. Instead of dividing data into blocks, Twister splits data into several files which allows users to take partitioned files as the input of Map function which gives users more flexibility. Due to the different methods adopted for handling input data with Hadoop and Twister, an additional process is required in Map class templates for allow the Map function in Twister only manipulate one line each time as well as Hadoop does.

A. Syntax Analysis

In order to analyze the Hadoop codes, we divide the codes into different levels. The first level includes three portions: **Package**, **Import Library**, and **MapReduce Class**. See Figure 2.

(I) "Package" can be copied directly.

(II) "Import Library" can be split into two smaller parts depending on whether the input code is a Hadoop library or not. If it is not a Hadoop library, the code is directly copied to translated codes. Otherwise, the codes are replaced with Twister import library template.

(III) "MapReduce Class" can be separated into five parts: **Map Class**, **Reduce Class**, **Combine Class**, **Main Function**, **and Others**. These five parts are included in second level of Figure 2 and the detailed method is given as follows.

- **Map Class** contains two smaller parts: "Map Function" represents the codes in the Map function, and "Others" are the codes out of the Map function, as shown in Level 3 of Figure 2.

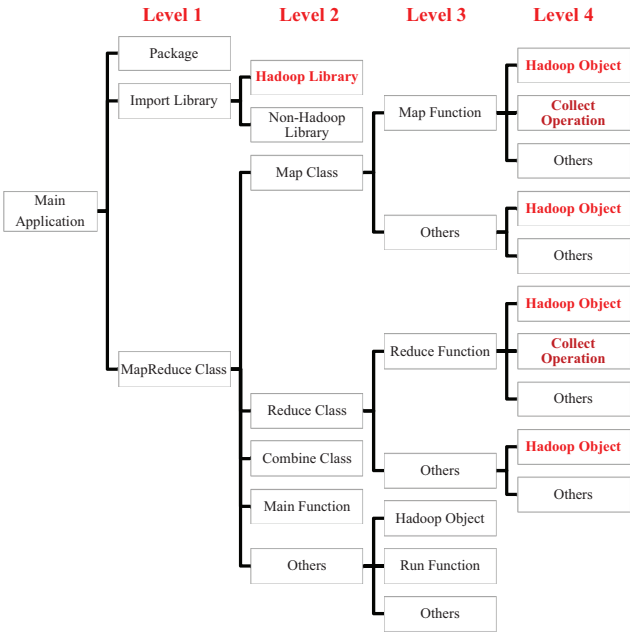

Figure 2. Syntax analysis

(a) In "Map Function", codes can be categorized as three segments: "Hadoop Object", "Collect Operation", and "Others" in Level 4 of Figure 2.

1) "Hadoop Object": if the code contains the Hadoop type object, we take it as a Hadoop Object that we should consider the different conditions of object declaration and function of object.
2) "Collect Operation": when the code contains ".write()" function, we assume it is a key/value pairs "Collect Operation" and replace it with the collect function of Twister.
3) "Others": the rest of codes in Map Function should be the naive JAVA codes that we can directly copy them to the translated code.

(b) "Others" of **Map Class** contains two segments: "Hadoop Object" and "Others" which we handle in the similar way as we do in "Map Function".

- **Reduce Class** has the similar structure with **Map Class**, we adopt the same strategy to analyze Reduce Class.
- **Combine Class** is an optional process after Map phase in Hadoop and not included in Twister. Thus, we do not deal with this part at present.
- **Main Function** is almost directly replaced with the generated templates with the small modifications which depend on the output key/value pairs of Reducers.
- **Run Function** is applied to execute MapReduce process and also replaced by a generated template for Twister.

B. Handle different data types

Different data types are used for representing various key/value pairs in both platforms. Thus, data type conversions between Hadoop and Twister is an important process during translation. In Hadoop, the same data type can be used for both the key and value. However, different data types are required in Twister for the type of key and value. For example, the data type *String* for key and value in Hadoop can be represented as a *Text* type for both the key and value, but in Twister, *StringKey* is required for the key of string type, and *StringValue* is utilized for value. Due to this difference, the translator is not able to directly convert Hadoop data types into Twister data types. At first, the translator recognizes the data type as key type or value type through the output type of key/value in Hadoop. Then, the translator converts it into the corresponding key type or value type in Twister.

Take **Map class** of Hadoop as an example, "Map Class" in Hadoop architecture should inherit the "Mapper Class" in the following format, "...extends Mapper<*LongWritable, Text, LongWritable, IntWritable*>". The first pair: <*LongWritable*, *Text*> is input data key and value type, and the second pair: <*LongWriable*, *IntWritable*> is the output type of key and value. In this way, the translator can decide that the variable with *LongWritable* type key should be translated to *LongKey* and the variable with *IntWritable* type value is translated to *IntValue* as the output key/value pair. Table II presents some representative data types in Hadoop and the corresponding key/value types in Twister. Here, we find that some key or value types do not exist, mark as "N/A". To deal with this situation, we can use "user-defined" type to replace them in H2T.

Table II
DATA TYPES

Hadoop Data Types	Twister Data Types	
Key/Value	Key	Value
Text	StringKey	StringValue
IntWritable	IntKey	IntValue
LongWriable	LongKey	N/A(user-defined)
NullWritable	N/A(user-defined)	NullValue
BytesWritable	N/A(user-defined)	BytesValue

C. Handle functions of Hadoop objects

Some specific functions of Hadoop objects should also be handled separately during the translation process. The current translator can only convert a few functions:

1) ".set()" function is used to assign a value to Hadoop type object such as TextObject.set(value). There is no similar function of Twister object can be exploited directly. Hence, we implement a new operation for translated Twitser code in order to achieve the same operation. Assume there is a string object for a key, the translated Twister code now can be "StringKey object = new StringKey(value);". (see line 4 in Table III)
2) "val.get()" is another function in Hadoop Reducer used to retrieve the value of IntWritable and ByteWritable objects; value.toString() is used for Text type object. In Twister, there are different functions to implement the processes. For IntValue, it has the value.getVal() function, and BytesValue has the value.getBytes() function. Conversion of these functions requires not only changing the functions name, but also typing casting depends on which value type the object is. (see line 3 in Table IV)
3) "Context.write()" in Hadoop is another operation used to collect key/value pairs for both the Mapper and Reducer as the output data. Twister needs two different operations for the map and reduce function, "MapOutputCollector.collect()" and "ReduceOutputCollector.collect()". For manipulating key/value collection process, codes can be simply altered from "context.write" to "collector.collect" if we utilize the same name for both collectors even though Twister has two different collectors. (see line 5 in Table III & line 4 in Table IV)

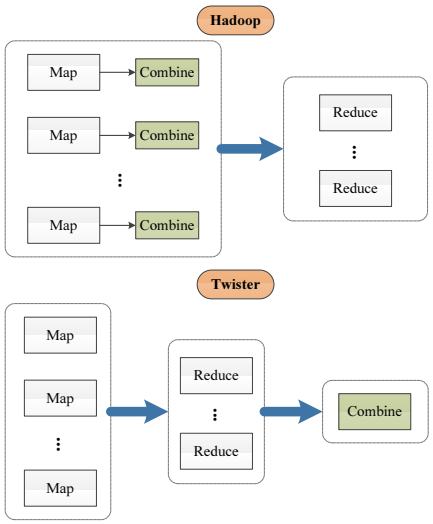

Figure 3. A comparison of flowchart in Hadoop and Twister

D. The Combine Class

Hadoop utilizes Combine process to merge output key/value pairs from Mapper for minimizing the size of intermediate output data. With the Combine class, communication loads between workers can be reduced significantly while processing the big data set. However, the main purpose of Combine class in Twister is to merge separated output data from different Reducers, and there is only one result generated. Figure 3 shows the difference of Combine in

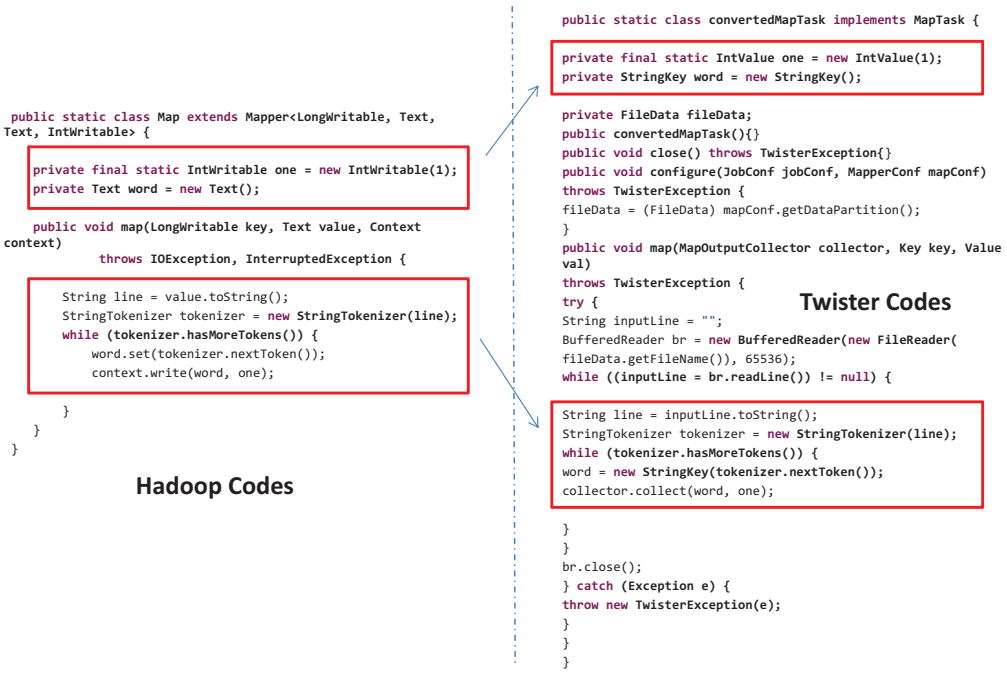

Figure 4. Converting Hadoop codes into Twister codes - Map

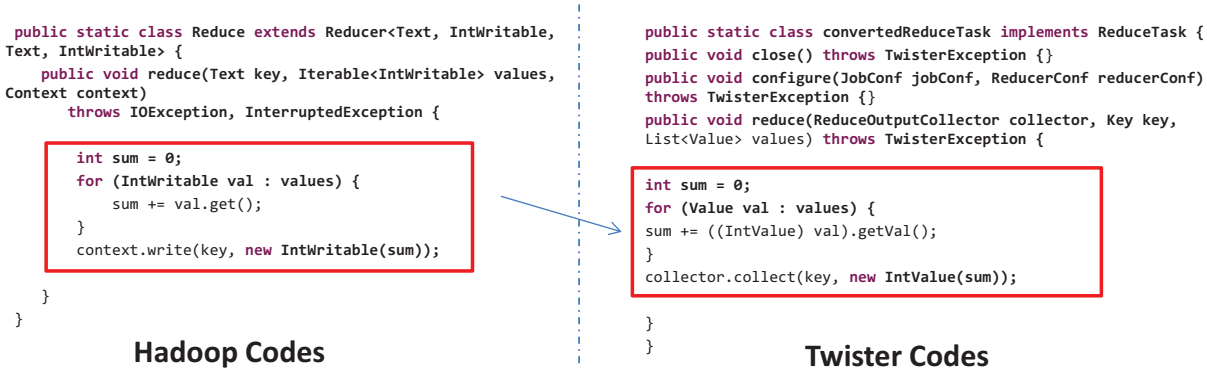

Figure 5. Converting Hadoop codes into Twister codes - Reduce

Hadoop and Twister. For the translation, we use a combine class template here for translated Twister code to collect the final result from reducers. The only thing needs to be modified for this template is the type of output key and value. Data types of the result should be changed depending on output key/value data types of Reducers.

E. The Run Function

A Template made for job configuration setting includes Map, Reduce and Combine setting as well as the number of map tasks and reduce tasks in **Run Function** of Hadoop codes. This function is called in main function for processing the MapReduce job, and returns the final result.

IV. An Illustration

In this section, we give an example on translating Hadoop codes to Twister codes using the WordCount application. The key step in translation is to translate the Map and Reduce functions. We extract Hadoop Map and Reduce functions through analyzing and recognizing "*extends Mapper*", "*extends Reducer*". Figures 4 and 5 show the Map and Reduce codes in Hadoop and Twister which both are JAVA-based. The Hadoop codes are copied/interpreted/replaced, then the Twister codes are generated. For analyzing the Hadoop Map and Reduce functions, we extract the input and output data types by using a regular expression "<*,*,*,*>". Then we set the corre-

Table III
COPIED OR INTERPRETED IN MAP FUNCTION

1	String line = value.toString();	interpreted	String line = inputLine.toString();
2	StringTokenizer tokenizer = new StringTokenizer(line);	copied	StringTokenizer tokenizer = new StringTokenizer(line);
3	while (tokenizer.hasMoreTokens()) {	copied	while (tokenizer.hasMoreTokens()) {
4	word.set(tokenizer.nextToken());	interpreted	StringKey word = new StringKey(tokenizer.nextToken());
5	context.write(word, one);}	interpreted	collector.collect(word, one);}

Table IV
COPIED OR INTERPRETED IN REDUCE FUNCTION

1	int sum = 0;	copied	int sum = 0;
2	for (IntWritable val : values) {	interpreted	for (Value val : values) {
3	sum += val.get();}	interpreted	sum += ((IntValue) val).getVal();}
4	context.write(key, new IntWritable(sum));	interpreted	collector.collect(key, new IntValue(sum));

sponding Twister key type and value type in Twister codes. The JAVA codes can be inherited, and we only translate the Hadoop data types to Twister data types. Besides, the special Twister codes are given directly. In more detail, we here propose which part is copied or interpreted in Map function and Reduce function, as shown in Tables III and IV, respectively. Take "Reduce function" as an example, according to different types and functions, the Twister Reduce codes in the box of Figure 5 are copied/interpreted. More clearly, this part of codes are listed in Table IV. It is easy to see that the first line is copied, and the following three lines are interpreted.

V. EXPERIMENTAL ANALYSIS

Our experiments are 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 in all, each has a clock frequency of 2.3 GHz). One of the nodes works as the master node and four others are worker nodes. The master node is not only responsible for initiating MapReduce applications, but also working as the host of message broker using Apache ActiveMQ. There are four daemons (one worker per daemon) running on each worker node in order to process the MapReduce job. The major mission of the daemon is to manage MapReduce tasks that assigned to it. The MapReduce runtime system exploits Twister 0.9 and JDK 1.7.0_05. A 200000×1000 matrix with the size of 933MB is randomly generated as the test data set for the our developed Hadoop programs MAX, MIN, MEAN, and SUM. WordCount data generator included in Twister 0.9 is applied to generate another test data set with the size of 1GB for the classical Hadoop program WordCount.

Figure 6 shows the comparison of hand-coded Hadoop applications, translated Twister applications, and hand-coded Twister applications. All the Hadoop applications contain the Combine process in order to merge the intermediate result generated by Map phase. However, the current translator does not support the Combine feature of Hadoop architecture yet. The translated Twister codes do not have the Merge process after Map phase, but we add this process later in the handed-coded Twister codes. As the result in Figure 7, we can see that even though the Hadoop applications apply the merge process, translated Twister applications still have the better performance. For the MEAN application, the Hadoop application and Twister application require almost the same time to process due to the large mount of generated intermediate results.

It is easy to see that hand-coded Twister applications have the best performance among these three different types of applications. Because of the merge process we exploit in hand-coded codes, the size of intermediate data is diminished significantly. In this condition, both communication time and load time are also reduced. Therefore, the hand-coded Twister applications perform the best.

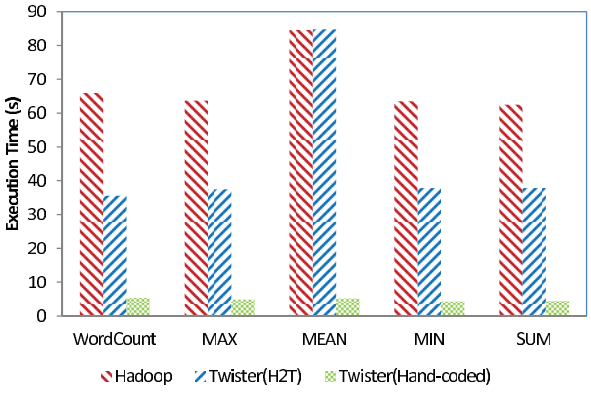

Figure 6. The Comparison of the Execution Time

VI. CONCLUSION

In this paper, we introduced a language translator for converting Hadoop codes into Twister codes in cloud systems, named H2T. The translator H2T can help users convert Hadoop-based applications to Twister-based applications which means "Write once, run two platforms". For all that, H2T is still in early stages at present. It can only convert some simple applications. One of our future work is to expand H2T to handle more complicated applications. Another

future work is to develop a translator to convert Hadoop codes into Twister4Azure codes where Twister4Azure is a Cloud version for Twister with C#-based language [14] and can help users run a Twister application on Windows Azure Cloud (https://www.windowsazure.com/).

ACKNOWLEDGEMENTS

This work is partially supported by the National Science Foundation of China (Nos. 61175047, 61100117, 61202043), the US National Science Foundation (No. OCI-1156733), the Fostering Foundation for the Excellent Ph.D. Dissertation of Southwest Jiaotong University 2012 and the Science and Technology Planning Project of Sichuan Province (No. 2012RZ0009), China.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] 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.

[3] J. Talbot, R. M. Yoo, and C. Kozyrakis, "Phoenix++: modular mapreduce for shared-memory systems," in *Proceedings of the second international workshop on MapReduce and its applications*, ser. MapReduce '11. New York, NY, USA: ACM, 2011, pp. 9–16.

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

[5] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a mapreduce framework on graphics processors," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 260–269.

[6] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 810–818.

[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, 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," *Proceedings of the VLDB Endowment*, 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 International Conference on*, 2011, pp. 25–36.

[10] J. Zhang, D. Xiang, T. Li, and Y. Pan, "M2m: A simple matlab-to-mapreduce translator for cloud computing," *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 1–9, 2013.

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

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

[13] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "The haloop approach to large-scale iterative data analysis," *The VLDB Journal*, vol. 21, no. 2, pp. 169–190, Apr. 2012.

[14] 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 International Conference on*, 2011, pp. 97–104.