MR-CloudSim: Designing and Implementing MapReduce Computing Model on CloudSim

Jongtack Jung and Hwangnam Kim
School of Electrical Engineering, Korea University
Seoul, Korea
{skylover89, hnkim}@korea.ac.kr

Abstract*—Nowadays, we can hear the word "cloud" everywhere. Cloud web storage, cloud-based web applications, and cloud networks, they all indicate that cloud has become more important than it has ever been. Internet service providers (ISP) and internet contents providers (ICP) are providing more and more services with cloud computing, and also in mobile computing environments, a cloud service is drawing much attention from the community. The biggest advantage of cloud computing is clear. As far as the user is connected to the cloud, (s)he can have all the extra computing power and storage no matter what his device is. However, empirical validation of various research projects on a cloud system seems to be ineffective in both cost and time since it requires involving to a huge amount of computing facilities and resources. Even with its importance as commerce and inconvenience of researching, there are not many simulators that can provide a simulation environment for cloud computing and datacenters. Thus, we propose a simulation tool that supports the *MapReduce* model, the most widely used programming model for big data processing, implemented on *CloudSim*, an open source simulator for cloud computing and datacenters.

***Keywords*— MapReduce, Cloud Computing, Performance Evaluation.**

I. Introduction

While more and more computing power is demanded every day, computing power of a single processing element has reached some point where it is practically impossible to make it any faster. This is the reason why even a mobile device has a multi-core CPU, which all tells us that loosely-coupled parallel computing has become a nature of recent computing technology.

For commercial servers processing petabytes of information every day, it requires a lot more than just a multi-core CPU. Because of this need for huge computing power, distributed computer network models have been adopted. A cloud network is a type of distributed computer network for large clusters of computers. Unless the data transferred per day and the need for computing power decreases, the importance of cloud networks will only be more highlighted.

However, contrast to its original purpose, cost efficiency, since a cloud network by definition requires a large cluster of computers, research and its validation on a cloud computing infra can be costly and possibly unavailable at all for those who cannot afford to build a cloud network of meaningful size. For these reasons, an empirical study with simulator supporting cloud computing is demanded, but there are not many of them yet [1][2].

One of the possible candidates is *CloudSim*[2]. *CloudSim* is yet incomplete. Its values are inaccurate, and it also lacks comprehension for a variety of software – many features in *CloudSim* do not have actual value or meaning. And probably because of above reasons, *CloudSim* is rather unknown. However, when the need for more computing power is ever-growing and both ISPs and ICPs are providing more and more services through clouds, the importance of cloud simulator will only grow. Note that the advantage of *CloudSim* is that it is an open source.

In this paper, we focus on *MapReduce*[4] model, one of the most commonly used computing model to exploit a cluster of servers. *MapReduce* is a dominating platform for big data processing. We then present an implementation of a bare bone structure of *MapReduce* on *CloudSim* environment. The goal of the implementation is to provide an easier and cheaper way to validate *MapReduce* operations with far less cost, given that *MapReduce* is usually even more demanding than others, as it is usually used for big data storage.

The remaining part of this paper is organized as follows. We explain the basic concepts and underlying simulator on which the proposed simulation framework grounds in Section II. In Section III, we describe a conceptual design on the proposed *MapReduce* simulation framework in *CloudSim* simulator, and then we explain how we implemented the proposed simulation framework in Section IV. We present the simulation results to show the correctness of the proposed simulation tool in Section V. Finally we conclude the paper with Section VI.

II. Preliminary

We would like to define some concepts used in the following sections before we proceed to introduce the proposed implementation of *MapReduce* on *CloudSim*.

A. Definition of a Cloud

The word "Cloud" has two meanings. The first, the more conventional, indicated the cloud computing service. This includes anything from web data storage service to web application to even virtualization of PCs. Vaguely, it means a user gets services from a "cloud" which symbolizes something they cannot perceive. More accurately, a cloud computing service is providing users with computing power and data storage of contents provider over the internet.

The word could also refer to a cloud network, which is less generally used because it is more technical. As mentioned before, a cloud network is a distributed computer network.

From here on, an ambiguous use of 'cloud' will be avoided, and either cloud computing or cloud network will be used in the paper.

B. MapReduce Model

MapReduce is a distributed computing model. Recently, there is a variety of engines adopting *MapReduce*. However, with their differences lying mostly in (if not entirely) external components such as schedulers, a bare bone *MapReduce* structure can be considered identical [5].

Inspired by *Map* and *Reduce* operations in functional programming languages such as Lisp and Pascal, *MapReduce* provides a functional style programming environment. The *master* node receives input files, sends them to the *slave* nodes, each node runs the file through *Map* operation, sends Map output files with keys to the intermediate stage, and collects the results of *Reduce* operations on each node [6].

MapReduce is suited for embarrassingly parallel or distributed problems, because its operations are completely independent on other files. In a commercial cloud network, loads (user applications or server applications such as indexing) are mostly independent on each other, making *MapReduce* an effective solution. The structure of MapReduce algorithm will be discussed as we describe the proposed model.

C. CloudSim

CloudSim is an open source simulator developed by CLOUDS lab in Melbourne Univeriosty. Beside *CloudSim* core functions, there are two main parts in a *CloudSim* simulation: entity and event [3].

1) Entity

Simulation entities represent the physical hardware part of a cloud network. Each entity can send/receive messages to/from other entities. In every simulation, entities are set up in the beginning and shut down at the end. Since entities represent hardware, creating an entity during a simulation is restricted.

Simulation entities are: *Datacenter, DatacenterBroker, CloudInformationService*, *NetDatacenterBroker*, and *Switch*.

2) Event

Since *CloudSim* is an event driven simulator, every incident is represented by an event. Every time an event is sent to an entity, the entity adds the event to the event queue of the *CloudSim* core. The queue is consistently sorted by the incident time of each event, and the *CloudSim* runs the events consecutively.

Although *CloudSim* only provides a skeleton structure of cloud networks, it covers wide range of features: processing, data storage, datacenter power, etc. While incomplete, *CloudSim* can give a good perspective to cloud networks. With right type of addition and modification, *CloudSim* can be a powerful tool to analyze any cloud service infra and validate any research on the infra.

III. DESIGN OF MAPREDUCE MODEL IN CLOUDSIM

We explain in this section how *MapReduce* model is designed and then it is incorporated into *CloudSim* while preserving the main characteristics of both of them.

A. MapReduce Model

MapReduce has only few notable features that we need to implement in its bare bone structure because most of the algorithm's intelligence lies in schedulers. Thus the features are as follows:

1. The workload may not be dependent on input file size.
2. Every separated Map output has one Reduce operation.
3. Reduce operation has to come after Map operation of corresponding input

The goal of *Map* operation can be two-fold: key assignment and intermediate record generation. The goal of *Reduce* operation is the final output generation. The goal of *MapReduce* as a whole is generating output file organizing input file as desired [6]. Since the method about how *Map* and *Reduce* operations generate their output file is user-defined, the workload of *MapReduce* may not depend on its input file size: note that it usually does.

Before going into reduce phase, each stage has the same number of files. For every input, there is a pair of *Map* and *Reduce* operation and no more. Recall that schedulers are not covered in this paper. Master node scatters and gathers data, but they are insignificantly small because of the nature of its work.

Third and last, *Reduce* operation always comes after *Map* operation of the same input. This is natural, considering how *MapReduce* works, but since a simulator does not actually run the program, it is good to note this to avoid any confusion.

B. Enhacements on CloudSim

Like the actual system, the *CloudSim* composes a cloud network with four major components: *datacenter*, *users*, *VM*s, and *cloudlet*s (workloads).

In *CloudSim*, the number of users and the list of VMs and Cloudlets are submitted to the datacenter. VMs are created when the datacenter is initialized, destroyed when all the cloudlets have been executed. Each user and cloudlets do not have direct correlation. So, suggested implementation ignores the *users* variable.

To provide a model with minimal unpredicted errors, we needed to implement the *MapReduce* model completely outside of the original *CloudSim* codes. Suggested *MapReduce* model mostly stands on its own.

However, to provide least compatibility, native *CloudSim* codes need some modification of the original code as well. The modifications are as follows: providing Datacenter class compatibility with implemented class, supplementary methods for Datacenterbroker class for implementation purposes. Details are elaborated in section IV.

IV. Implementation of MapReduce Model on CloudSim

As mentioned before, the purpose of the implementation is to provide an easier and cheaper way to examine *MapReduce* model.

In this implementation, we replace some native classes with implemented module. It provides convenience in variable manipulation to use custom class, but it can also cause unpredicted error. The goal of this paper is to provide easier way to examine MapReduce model, so this kind of problem has to be avoided.

The implementation is composed of two parts based on aforementioned design decisions: modification and addition. Modifications are made on native *CloudSim* code, including Datacenter and DatacenterBroker. And the entire *MapReduce* model has been added. Figure 1 describes an execution flow of one simulation scenario passing through *CloudSim* code and newly appended code for *MapReduce* model on *CloudSim*.

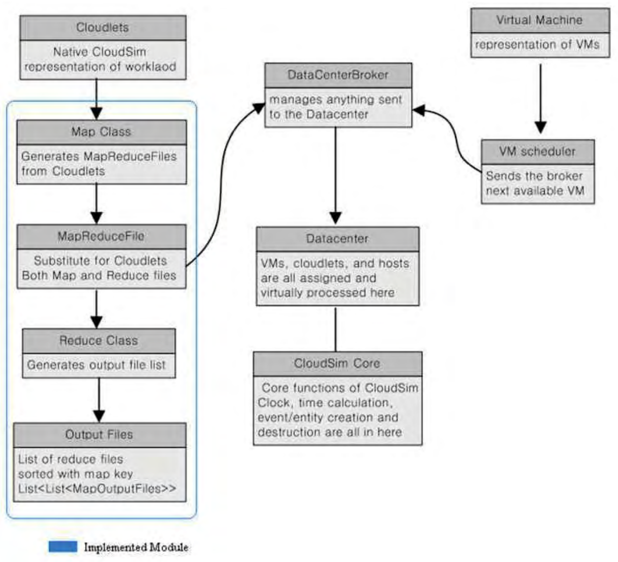

Cloudlets
Native CloudSim representation of workload
Map Class
Generates MapReduceFiles from Cloudlets
MapReduceFile
Substitute for Cloudlets Both Map and Reduce files
Reduce Class
Generates output file list
Output Files
List of reduce files sorted with map key List<List<MapOutputFiles>>
DataCenterBroker
manages anything sent to the Datacenter
Datacenter
VMs, cloudlets, and hosts are all assigned and virtually processed here
CloudSim Core
Core functions of CloudSim Clock, time calculation, event/entity creation and destruction are all in here
Virtual Machine
representation of VMs
VM scheduler
Sends the broker next available VM
Implemented Module

FIGURE 1. AN EXECUTION FLOW OF MR-CLOUDSIM

A. *MapReduce Model*

In simulation, there is no user-defined function, no actual input files; though *CloudSim* does provide a toolkit for actual file processing, purpose of simulation, efficiency in time and cost, is lost by doing so. Virtual data and workload are sent and processed, which makes the operation for key generation unnecessary. The basic structure of implemented model is as below.

1. Generate a cloudlet and run it with Map class.
2. Map class creates MapReduceFiles based on cloudlet attributes and size of a map file.
3. Created files are marked as “map” files, and keys are assigned manually.
4. Each “map” output file generates a “reduce” file, and they are “paired.”
5. Both “map” file and “reduce” file are sent to pre-set intermediate file list. This list replaces cloudlet list
6. Intermediate file list – MapReduceFile list – is submitted to the broker. The broker only passes “map” files to the datacenter.
7. Whenever a “map” operation is finished, submit the corresponding “reduce” operation to the datacenter.

Although not utilized, at each stage the file size of cloudlets is adjusted accordingly. The list of classes implemented is as follows: Aggregator, Map, MapReduceFile, MapReducePair, and Reduce.

1) *Aggregator*

Aggregator class represents list of intermediate files. In an actual system, the intermediate files would be sent to the master node for the reassignment to another VM. However in *CloudSim*, scheduler is externally defined. Thus an aggregator is required.

Since every single cloudlet needs to be submitted all together, even if there are multiple map classes, there should be only one aggregator. *MapReduce* classes all work based on the aggregator, so an aggregator needs to be initialized before any other classes.

2) *Map*

While seperating input file into segments and scattering them is master node’s job, in this simulation, it is done in the Map class. Map class accepts a cloudlet, and separates it into a number of MapReduceFiles.

These files are marked as “map” file, and the class creates a “reduce” file for every “map” file. These two are paired using the class MapReducePair.

Since file size attribute of a cloudlet is easily overlooked and set to be an unrealistic number, Mapping here varies on the workload of a cloudlet. It is more realistic to use file size, so a module for that is also provided.

Keys for the map files are assigned in this class. Again, generating keys depending on input file contents does not suit the purpose – once again, it is possible –, so they are manually assigned. In this paper, keys for MapReduceFiles of a cloudlet are numbered consecutively as they get separated. Assigning the distribution of keys could be one of applications.

3) *MapReduceFile*

This class is defined to provide cloudlet with essential characteristics for the input of the *MapReduce* and additional features for the implementation.

To make our *MapReduce* implementation work in native *CloudSim* modules, MapReduceFile inherits Cloudlet class so that a programmer only has to modify target attributes of each entity.

Additional features of MapReduceFile include: key, map/reduce marker, and pair parent pointer. Key is the essential characteristic for MapReduce algorithm, and the other two are for implementation purpose.

4) MapReducePair

This class is made for an easier programming experience. A Map and a Reduce tend to relate to each other very often, so binding them as a pair could save a lot of time.

5) Reduce

Reduce class does not work along with the *CloudSim* core functions. The Reduce class accesses the target Aggregator and sorts only the "reduce" marked MapReduceFiles according to their keys.

Since reduce functions in most platforms providing MapReduce generate output by assembling the map output files with the same key, pre-set method sorts the files in the same way. Output files are stored in reduce class and can be accessed if needed.

B. Native Class Modifications

1) DatacenterBroker

A broker receives available VM information from the datacenter and submits cloudlets to the VM.

Since *CloudSim* requires every cloudlet to be listed in cloudlet list submitted to Datacenter in the beginning, Datacenter broker has to delay *Reduce* from starting before corresponding *Map* has ended.

2) Datacenter

Since a *Reduce* operation can only start after corresponding *Map* operation, it is important to know the time when a Map operation finishes.

Because broker does not have the full information for the time, the job of Reduce assigning has to be done in Datacenter class. Thus, Datacenter is modified to send Reduce job back to broker whenever a Map job has been finished.

V. PERFORMANCE EVALUATION

In this section, we present the simulation results that the proposed *CloudSim* produced. In this simulation, SpaceShared scheduler was used, because it is the realistic type of scheduler; SpaceShared scheduler is the native scheduler that assigns VMs one cloudlet at a time.

A. Simulation Parameters

There are three components in *CloudSim* that the user has to define for simulation: VM, workload, datacenter. Below (Table I) are parameters that are going to be consistent throughout the entire evaluation.

TABLE I. DATACENTER PARAMETERS

Number of PE	4
Ram	16384
Storage	1000000
Bandwidth	10000
MIPS	1000

Although Datacenter Class requires a few other parameters such as OS, system architecture, and VMM to be defined for initialization, they are irrelevant. *CloudSim* has many features like these, and those will not be mentioned from here on.

TABLE II. VM PARAMETERS

Number of PE	2
Ram	512
Image Size	10000
bandwidth	1000
MIPS(standard)	250

In *CloudSim*, VM list needs to be submitted ahead of simulation. Any VM missing from the list will be considered non-existing to *CloudSim* core. In the same vein, any failure of VM creation does not change the elements in submitted VM list. Unless the broker is supplemented with a VM failure monitor, VM parameters need to be monitored by the programmer. That means the each sum of VM Ram, Image Size, bandwidth, or MIPS needs to be less than the corresponding Datacenter parameter. The parameters used in this performance evaluation are described in Table II.

MIPS of VMs is randomly set to 1, 1/2, and 1/4 of standard MIPS to verity if Reduce is properly deployed. Any other parameter is consistent.

B. Methodology of Validation

To validate the implemented model, we are going to run four simulations: uniform distribution of Reduce output file sizes, sorting status of Reduce output files, representation of physical attribute, and verification of compatibility with the rest of *CloudSim*. In this section, only methodology is going to be introduced. We will call each simulations simulation 1, 2, 3, and 4 respectively. Detailed simulation procedure and exact parameters will be given in section C. Note that while entire Map operation is only assigned to one node, in all simulations of the paper assigns each separated Map output to a node to provide better demonstration.

Validation of uniform distribution of Reduce output, which is simulation 1, is simple. A small number of cloudlets will be created and processed through Map class and generate MapReduceFiles, and processed through to the end. Map key is irrelevant to the purpose of simulation, so the key generation is left untouched. After that, print out the characteristics of MapReduceFiles with 'reduce' mark pulled directly out of Aggregator. In this simulation, identical size of reduce files with different ID will show that Mapping has been properly done. The number of cloudlets is kept small because otherwise the result will consume too much space.

Sorting status verification, which is simulation 2, will be done by showing Reduce output files listed with each index with their keys. To reflect actual systems, sorting of output files are done so that the files with the same map key are listed together. Since the consecutive key assignment of native implementation can be incomprehensive, random integer key is assigned in this simulation.

In simulation 3, we are going to show that Reduce operation always comes after corresponding Map operation. This is by far the most important feature because it represents the physical trait of *MapReduce* model. Every Reduce operation can only happen with the key and value pair generated by corresponding Map operation [2].

For this simulation, we utilize two properties of Cloudlet class: CloudletExecTime and CloudletFinishTime. In *CloudSim* core, every listed cloudlet is processed, and the time it starts and finishes are recorded in CloudletExecTime and CloudletFinishTime, respectively. By subtracting starting time of Reduce file and finishing time of corresponding Map file, we can visualize the result. We will call this the *Delay Time*. Of course, if the *Delay Time* is negative, the module is not working properly.

Just as a demonstration of compatibility of implemented module to the rest of *CloudSim*, we also added Simulation 4. Simulation 4 verifies compatibility of implemented module by running *CloudSimExample8* simulation [citation] with little modification. Modifications made are limited to: addition of Map class, replacement of Cloudlets by MapReduceFiles, removal of GlobalDatacenter. This simulation will show that the implemented module will create no unpredicted error in *CloudSim*.

C. Simualtion Results

1) Simulation 1 – uniform distribution validation.

The purpose of simulation 1 is to show output files of implemented *MapReduce* model are properly distributed uniformly. Input parameters are presented in Table III.

TABLE III. INPUT FILE PARAMETERS FOR SIMULATION 1

Load (standard)	40000
File size (standard)	3000
Map size	10000
# of Cloudlets	3

To provide generality in this simulation, load and file size of each cloudlet is multiplied by a random number. Total number of initial cloudlets is 3, limited to small number because of the reason mentioned above. The resulting file sizes of each initial cloudlets are: 6000, 6000, and 3000.

TABLE IV. OUTPUT FILE GENUINENESS (SIMULATION 1)

ID	Workload	File Size	Output Index
0	10000+100	750	1
1	10000+100	750	1
2	10000+100	750	1
0	10000+100	750	2
1	10000+100	750	2
2	10000+100	750	2
0	10000+100	750	3
1	10000+100	750	3
2	10000+100	750	3
0	10000+100	750	4
1	10000+100	750	4
2	10000+100	750	4
0	10000+100	750	5
1	10000+100	750	5
0	10000+100	750	6
1	10000+100	750	6
0	10000+100	750	7
1	10000+100	750	7
0	10000+100	750	8
1	10000+100	750	8

In this simulation, since file size is manually managed by us, implemented module uses file size to separate each Map files. As seen above, each file size and workload is uniformly divided into segments. As of workload notation, the former is Map size and the latter is Reduce size, which is kept 1/100 if not specified.

There are a total of 20 output segments. File sizes add up to 15000, and this equals to the sum of initial cloudlet file size, which indicate that there has been no additional or left out segment. Table IV presents the specific results.

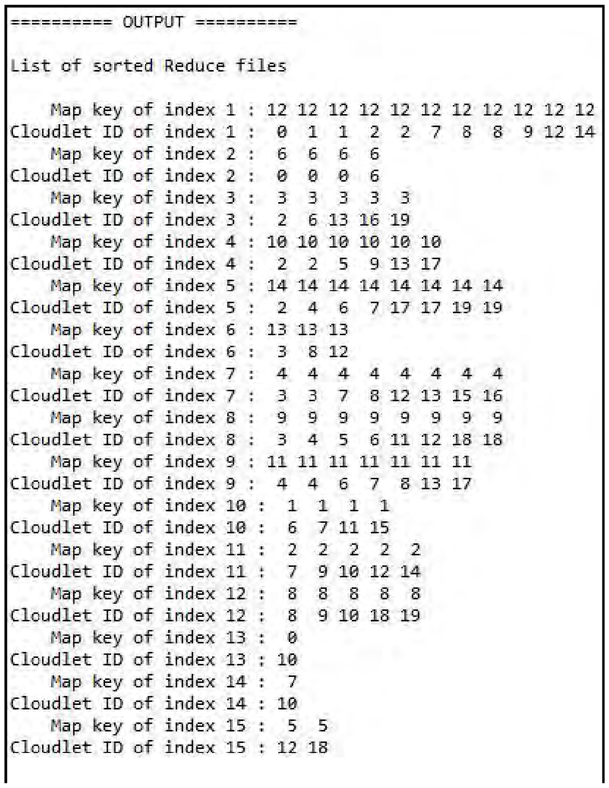
2) Simulation 2- output file genuineness

The purpose of simulation 2 is to verify that each output file with different keys is properly sorted. Every file with the same key has to be indexed together, and any two files with different keys must be indexed apart. Table V is initial cloudlet parameter for simulation 2.

TABLE V. INPUT FILE PARAMETERS FOR SIMULATION 2

Load (standard)	40000
File size (standard)	3000
Map size	20000
# of Cloudlets	3

The result pool of simulation 2 is even smaller than simulation 1. We assumed this would be enough for the demonstration.



```
========== OUTPUT ==========

List of sorted Reduce files

    Map key of index 1 : 12 12 12 12 12 12 12 12 12 12 12
Cloudlet ID of index 1 :  0  1  1  2  2  7  8  8  9 12 14
    Map key of index 2 :  6  6  6  6
Cloudlet ID of index 2 :  0  0  0  6
    Map key of index 3 :  3  3  3  3  3
Cloudlet ID of index 3 :  2  6 13 16 19
    Map key of index 4 : 10 10 10 10 10 10
Cloudlet ID of index 4 :  2  2  5  9 13 17
    Map key of index 5 : 14 14 14 14 14 14 14 14
Cloudlet ID of index 5 :  2  4  6  7 17 17 19 19
    Map key of index 6 : 13 13 13
Cloudlet ID of index 6 :  3  8 12
    Map key of index 7 :  4  4  4  4  4  4  4  4
Cloudlet ID of index 7 :  3  3  7  8 12 13 15 16
    Map key of index 8 :  9  9  9  9  9  9  9  9
Cloudlet ID of index 8 :  3  4  5  6 11 12 18 18
    Map key of index 9 : 11 11 11 11 11 11 11
Cloudlet ID of index 9 :  4  4  6  7  8 13 17
    Map key of index 10 :  1  1  1  1
Cloudlet ID of index 10 :  6  7 11 15
    Map key of index 11 :  2  2  2  2  2
Cloudlet ID of index 11 :  7  9 10 12 14
    Map key of index 12 :  8  8  8  8  8
Cloudlet ID of index 12 :  8  9 10 18 19
    Map key of index 13 :  0
Cloudlet ID of index 13 : 10
    Map key of index 14 :  7
Cloudlet ID of index 14 : 10
    Map key of index 15 :  5  5
Cloudlet ID of index 15 : 12 18
```

FIGURE 2. SIMULATION 2 RESULTS

As shown in Figure 2, any index has only identical map keys. The result is aligned in a way that map key and cloudlet id with the same index and column is a pair to indicate the same output file. Output sorting does not depend at all on cloudlet id.

D. *Simulation 3 – Physical attribute representation*

In simulation 3, we only provided distribution of the *time delay*, so it was possible to provide a larger number of cloudlets. Under the impression that more data are always better, we implemented more cloudlets here. Table VI enumerates the specific parameters for this simulation, and Table VII shows the distribution of time delay measured in this simulation.

TABLE VI. INPUT FILE PARAMETERS FOR SIMULATION 3

Load (standard)	40000
File size (standard)	3000
Map size	10000
# of Cloudlets	20

TABLE VII. DISTRIBUTION OF TIME DELAY (SIMULATION 3):

Delay (100s)	~0	0	~1	~2	~3	~4
	0	44	4	4	2	4
Delay (100s)	~5	~6	~7	~8	~9	~10
	6	2	5	12	12	8
Delay (100s)	~11	~12	~13	~14	~15	~16
	12	8	9	2	2	4
Delay (100s)	~17	~18	~19	~20	~21	~22
	2	2	4	2	2	4
Delay (100s)	~23	~24	~25	~26	26~	
	2	2	2	2	0	

Unit in Table VII is in 100s. As the simulation is meant to be, the time delay is always 0 or greater. Because 0 was the most significant time delay, we separately added a column for 0. The reason why *time delay* is mostly 0 is because whenever there is any VM available, Reduce is instantly allocated to the VM right after corresponding Map is done.

E. *Simulation 4 – Compatibility check*

The purpose of simulation 4 is to verify that the module is compatible with the rest of *CloudSim* without any extra codes. Beyond that purpose, the result also provides a simple supplementary demonstration of simulation 3, representation of physical attributes. Table IIX and IX describe the simulation parameters and results, respectively. In Table IX, Reduce always starts after Map of the same key has finished.

TABLE IIX. INPUT FILE PARAMETERS FOR SIMULATION 4

Load (standard)	40000
File size (standard)	3000
Map size	20000
# of Cloudlets	5

VI. CONCLUSION

In this paper, we have implemented the *MapReduce* model in *CloudSim* and verified it with a simulation study. Since *CloudSim* does not provide much perspective in the file size and the contents of the files, implementing *MapReduce* model requires to modify and/or imporve existing functions of *CloudSim*. We made in this research some adjustments and assumptions to implement *MapReduce*. Note that we focused on implementation of bare bone structure of *MapReduce*. Even though the implemented system cannot represent the real world cloud systems by itself, the resulting system still provides better perspective at *MapReduce* computing model. In overall, a popular distributed computing model *MapReduce* is successfully implemented on the simulator *CloudSim*. We envision that this work can provide an easier way to examine *MapReduce* model in a datacenter.

As future work, we have several research directions. To make it more realistic, we would like to elaborate the implemented Mapreduce simulation framework. We also plan to incorporate the various network protocols and systems into the model in order to present the effects of the underlying network systems on the cloud system. And lastly, we will conduct a comparison study between the prospective complete MapReduce simulation framework and the real system such as Hadoop [6].

TABLE IX. SIMULATION 4 RESULTS

ID	STATUS	VM ID	Time	Started	Finished
000	Map	0	20	0.1	20.1
200	Map	3	20	0.1	20.1
201	Map	0	20	20.1	40.1
101	Map	1	40	0.1	40.1
100	Map	4	40	0.1	40.1
401	Map	3	20	20.1	40.1
100	Reduce	0	20	40.1	60.1
400	Map	1	40	40.1	80.1
001	Map	2	80	0.1	80.1
301	Map	4	40	40.1	80.1
301	Reduce	0	20	80.1	100.1
001	Reduce	3	20	80.1	100.1
000	Reduce	1	40	80.1	120.1
200	Reduce	4	40	80.1	120.1
201	Reduce	1	40	120.1	160.1
300	Map	2	80	80.1	160.1
401	Reduce	4	40	120.1	160.1
300	Reduce	3	20	160.1	180.1
101	Reduce	2	80	160.1	240.1
400	Reduce	2	80	240.1	320.1

ACKNOWLEDGMENT

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0014060) and in part supported by the grant from the Ministry of Knowledge Economy(MKE) of Korea (No. 10041120).

REFERENCES

[1] Rajkumar Buyya, Manzur Murshed "GridSim: a toolkit for the modeling and sumlation of distributed resource management and scheduling for Grid computing," University of Melbourne, 6 Jan 2003

[2] Rodrigo N. Calheiros, Rajiv Ranjan, Cesar A. F. De Rose, and Rajkumar Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," The University of Melbourne, Australia

[3] Rodrigo N. Calheiros, Rajiv Ranjan, Cesar A. F. De Rose, and Rajkumar Buyya, "CloudSim: a toolkit for modeling and simulation of cloiud computing environments and evaluation of resource provisioning algorithms," The University of Melbourne, Australia

[4] Jeffery Dean, Sanjay Ghemawat "MapReduce: Simplified Data Processing on Large Clusters," Google, Inc. a

[5] Matei Zahara, Andy Konwinski, Anthony D. Joseph, Randy Katz, Ion Stoica "Improving MapReduce performance in heterogeneous environments," USENIX'08 conference

[6] http://hadoop.apache.org/common/docs/r0.17.0/mapred_tutorial.htm