Portable Map-Reduce Utility for MIT SuperCloud Enviornment

Chansup Byun, Jeremy Kepner, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Matthew Hubbell, Peter Michaleas, Julie Mullen, Andrew Prout, Albert Reuther, Antonio Rosa, Charles Yee
MIT Lincoln Laboratory, Lexington, MA, U.S.A

***Abstract*— The MIT Map-Reduce utility has been developed and deployed on the MIT SuperCloud to support scientists and engineers at MIT Lincoln Laboratory. With the MIT Map-Reduce utility, users can deploy their applications quickly onto the MIT SuperCloud infrastructure. The MIT Map-Reduce utility can work with any applications without the need for any modifications. For improved performance, the MIT Map-Reduce utility provides an option to consolidate multiple input data files per compute task as a single stream of input with minimal changes to the target application. This enables users to reduce the computational overhead associated with the cost of multiple application starting up when dealing with more than one piece of input data per compute task. With a small change in a sample MATLAB image processing application, we have observed approximately 12x speed up by reducing the application startup overhead. Currently the MIT Map-Reduce utility can work with several schedulers such as SLURM, Grid Engine and LSF.**

Keywords—MIT Map Reduce; performance; scheduler; Grid Engine; SLURM; LSF

I. Introduction

Rapidly increasing data volume, velocity and variety has created a growing gap between data and users. This is true for the scientists and engineers at MIT Lincoln Laboratory as well. The common big data architecture, which is designed to address these challenges, is made of the computing resources, scheduler, central storage file system, databases, analytics software and web interfaces [1]. These components are common to many big data and supercomputing systems. The platform is designed to support standardized data access and dynamic compositions of functionalities.

In particular, addressing data volume requires a large computing cloud. The MIT SuperCloud [1] has evolved to merge the four common cloud computing ecosystems, namely enterprise, compute, database and big data clouds. In big data cloud computing, the open-source Map Reduce programming model is a very popular and widely used tool first described in 2004 by Google [2]. The open source community has its own implementations such as Hadoop MapReduce framework [3].

Although its underlying concept has existed in other programming models such as map and reduce primitives in Lisp and many other functional languages [3], Map Reduce programing became popular with the Hadoop MapReduce framework for the Java community. The Map Reduce programming model provides a number of benefits such as automatic parallelization and fault-tolerant features for Java programmers [3]. However, although the support has been extended to other languages such as Python [4], it still requires a steep learning curve for programmers who are not familiar with the framework.

In addition, scientists and engineers at MIT Lincoln Laboratory must work with legacy codes which may not be written in Java or Python. So we developed and deployed the MIT Map-Reduce utility to MIT SuperCloud systems [5], which works on a central storage system instead of distributed filesystem such as Hadoop distributed filesystem (HDFS) [6]. The Map-Reduce utility can launch any program onto the MIT SuperCloud with the use of the scheduler running on it. It can distribute the workloads in a block or cyclic distribution fashion. Since the initial deployment, the utility has evolved with more features. One of the new features is that it can reduce the runtime by consolidating the multiple launches of the mapper application into a single application launch per each compute task. This requires the modification of the mapper application so that it can process the input stream automatically generated by the utility. With the input consolidation, we have observed more than 2x speedup with toy examples and approximately 12x speedup for a user application. Currently, the MIT Map-Reduce utility can work with the majority of schedulers such as SLURM [7], open and commercial distribution of Grid Engine [8, 9, 10] and IBM Platform LSF [11]. However, the utility was written with the support for a wide range of schedulers in mind; it is reasonably trivial to add support for any other schedulers.

II. Portable map Reduce Utility

The Map-Reduce parallel programming model is the simplest of all parallel programming models; it is much easier to learn than message passing or distributed arrays. The Map-Reduce parallel programming model consists of two user written programs: Mapper and Reducer. The input to Mapper is a file and the output is another file. The input to Reducer is the set of Mapper output files. The output of Reducer is a single file. Launching consists of starting many Mapper programs each with a different input file. When the Mapper programs all have completed, the Reduce program is run on the Mapper outputs.

The MIT Map-Reduce utility has been deployed on the MIT SuperCloud systems by utilizing a centralized high-performance parallel filesystem such as the Lustre filesystem

Distribution A: Public Release. This work is sponsored by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.

[12]. Since any high-performance supercomputing facility runs a scheduling or resource management software, the MIT Map-Reduce utility is designed to use the existing scheduler to manage its workloads. Finally, the MIT Map-Reduce utility assumes that users will have their data already partitioned into many smaller segments. Such segmentation is natural for many application areas; when collecting data from various sensors, they are collected in a large number of segmented files instead of one large, holistic file. This allows users to deploy their applications rapidly and efficiently.

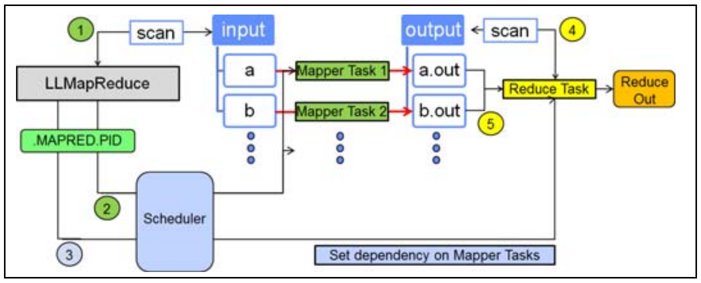

Fig. 1. A schematic diagram showing how MIT Map-Reduce works. The Map Reduce process identifies input files which are used to generate an array job with the help of the HPC scheduler. By setting a dependency between the mapper and reducer jobs, the output of completed jobs are passed through a reducer to generate the final result.

The MIT Map-Reduce utility, called LLMapReduce, identifies the input files to be processed by scanning a given input directory or reading a list from a given input file as shown in the step 1 in Fig. 1. It generates all the necessary temporary files under the directory, .MAPRED.PID, where the PID is the process identification number of LLMapReduce process in which it was executed.This new feature has been added so that users can build a nested call to LLMapReduce for their hierarchical data processing construction. Then, by accessing the scheduler at step 2, it creates an array of many tasks, called an array job, which is denoted as “Mapper Task 1”, “Mapper Task 2”, and so on. LLMapReduce was originally written to work with the open source Grid Engine [9, 10], and more recently it has been extended to work with SLURM [8] and LSF [11] as well.

Once the array job is created and dispatched for execution, each input file will be processed by one of the tasks with the specified application at the command line, noted as “Mapper” in Fig. 1. The application can be any type of executable written in any language, such as a shell script, a Java program or a MATLAB script. In addition, there is an option to do further processing on the results, if there are any, by creating a dependent task at the step 3. This is noted as “Reduce Task” in Fig. 1. The reduce task will wait until all the mapper tasks are completed by setting a job dependency between the mapper tasks and reduce task. The reduce application is responsible to scan the output files from the mapper tasks at step 4 and to process them into the final results at step 5.

The map application of the MIT Map-Reduce utility requires two input arguments: one for the input filename and the other for the output filename. Subsequently, the reducee application takes two arguments as input, which are the directory path where the results of the map tasks reside and the output filename for the reduce result. The reduce application may use the input path to scan and read the output generated by the map tasks.

```
LLMapReduce --np=Np                    \
            --input=input_dir          \
            --output=output_dir        \
            --mapper=myMapper          \
            --reducer=myReducer        \
            --redout=output_filename   \
            --ndata=NdataPerTask       \
         --distribution=block|cyclic   \
            --subdir=true|false        \
            --ext=myExt                \
          --delimeter=myExtDelimiter   \
            --exclusive=true|false     \
            --keep=true|false          \
            --apptype=mimo|siso        \
  --options=<scheduler_options_to_add>
```

Fig. 2. Available options of MIT Map-Reduce utility.

The available options of the current MIT Map-Reduce utility are shown in Fig. 2. After we deployed the utility initially, we found out that some users tried to launch their applications with the number of data files exceeding the limit that the scheduler array job could accommodate. So we modified the --np option in such a way that it not only limits the total number of compute tasks it generates, but also calculates the number of data files per each task to be assigned. The --ndata option allows users to define how many data files are to be assigned per each task, which will override the --np option. The --ext option allows changing the default extension, “out”, with the user-defined extension name. Along with the --ext option, the --delimiter option allows the change of the default, “.” extension with a user-defined delimiter when adding the extension. The --distribution option enables changing how the input data is distributed among the given number of task processes; the default is the block distribution. The --subdir option is useful if your data files are stored in a hierarchical directory structure. By defining this option with the top directory path of your input data, the utility will traverse all the sub-directories to process all the data files. The --exclusive option enables the use of entire compute nodes for your jobs, but this option is limited to pre-approved users as configured in the current Grid Engine scheduler running on the MIT SuperCloud systems. By default, the utility will delete the .MAPRED.PID directory after the job is completed. However, users can keep the temporary directory for debugging purpose with the --keep=true option. By default, the utility expects that the map application to take single input and single output path (siso) at a time. However, this will incur overhead associated with repeated startups of the map application.We have observed that some applications such as MATLAB codes can save significant overhead cost with the minor change of having the map application start only once and read many lines of input/output path pairs to process the given data. For this purpose the --apptype=mimo option will generate the input files for the modified map application that will read the input file with the multiple lines of input/output filename pairs.

Finally, the utility allows adding some additional scheduler options when generating the job submission scripts with the --options option. This is handy when some data processing requires more memory than the standard allowance.

III. Use Cases

In this section, we present a couple of use cases to demonstrate how to use the utility. First, a MATLAB application that converts an RGB image into a gray-scale image is used with a small number of image files. It demonstrates the use of the Mapper and assigning multiple input files to one application execution. Next, a Java application that counts the number of unique words in the given text files illustrates the use of the Mapper and Reducer.

A. A MATLAB Application

An image conversion function, called imageConvert(), is shown in Fig. 3. The function takes two arguments, the input and output image names. It reads in an RGB image file and converts it into a gray scale image. Then, it saves the gray scale image into the file of the output name. This satisfies the MIT Map-Reduce API requirements.

```
function imageConvert(inFile,outFile)
I=imread(inFile); J=rgb2gray(I);
dicomwrite(J,outFile);
```

Fig. 3. A MATLAB application that convert an image file from RGB to gray scale.

However, it still needs a wrapper script to receive the input and output file names that are provided by the utility. Then, the wrapper script will execute MATLAB with the corresponding input and output files when dispatched for execution by the scheduler. An example wrapper script is shown in Fig. 4.

```
#!/bin/bash
cat<<EOF|matlab -nodisplay -singleCompThread
inFile='$1'; outFile='$2';
imageConvert(inFile, outFile);
EOF
```

Fig. 4. An example wrapper script, MatlabCmd.sh, for the imageConvert() function.

In the above wrapper script, the variables, $1 and $2, are the two input arguments (input and output file names), which are provided by the MIT Map-Reduce utility. The script will execute the imageConvert() function with MATLAB when it is called by the run script, which is generated by the utility. With the wrapper script, the image conversion job can be launched with one line of the MIT Map-Reduce command as shown in Fig. 5. In this case, each input image file in the input directory becomes a compute task of an array job automatically generated by the utility. The resulting gray images are saved in the output directory as specified.

When the MIT Map Reduce command is called, some temporary files are created in the .MAPRED.PID directory,

```
$ LLMapReduce –-mapper MatlabCmd.sh \
              --input input –-output output
```

Fig. 5. An example Map Reduce job with the MIT Map-Reduce utility.

where PID is the process identification (PID) number of the MIT Map-Reduce command. These temporary files are generated for the specific scheduler being used on a particular MIT SuperCloud system. The files are one job submission script and a number of run scripts for all compute tasks, one per each compute task as shown in Figs. 6 and 7, respectively. The job submission script shown in Fig. 6 is written for the open source Grid Engine scheduler, which has a number options specific to the scheduler. The -t 1-M option specifies an array job of M tasks, starting from 1 to M with an increment of one. The number M is determined by the MIT Map-Reduce utility, which is the number of input image files in the input directory. Each compute task keeps its own log file, uniquely named with its job and task numbers. If there is any standard output, it goes into these log files.

```
#!/bin/bash
#$ -terse –cwd –V -j y -N MatlabCmd.sh
#$ -l excl=false -t 1-M
#$ -o .MAPRED.1120/llmap.log-$JOB_ID-$TASK_ID
./.MAPRED.1120/run_llmap_$SGE_TASK_ID
```

Fig. 6. An example job submission script written for the Grid Engine scheduler.

```
$ cat .MAPRED.1120/run_llmap_1 (for task 1)
#!/bin/bash
export PATH=${PATH}:.
MatlabCmd.sh input/image_1.jpg \
             output/image_1.jpg.out
. . .

$ cat .MAPRED.1120/run_llmap_M (for task M)
#!/bin/bash
export PATH=${PATH}:.
MatlabCmd.sh input/image_M.jpg \
             output/image_M.jpg.out
```

Fig. 7. A number of run scripts for all compute tasks generated by the MIT Map-Reduce utility.

The run script for each compute task is written to feed one input and one output argument to the wrapper script shown in Fig. 4. This meets the MIT Map-Reduce Application Programming Interface (API) requirement. As mentioned above, the MIT Map-Reduce utility generates M number of run scripts, one run script per each compute task. As shown in Fig. 7, the output file name is determined by the name of the input file with the default extension, “.out”.

However, the example shown in Fig. 5 has an issue if there are a large number of input files in the given input directory. As discussed earlier, we have observed that some users tried to launch a job with more than 100,000 data files. This can easily break the scheduler limit for how many tasks a job array can have. For example, the default maximum number of tasks of an array job is 75,000 for the open source Grid Engine scheduler. In order to handle this case, the --np option can be used to specify how many compute tasks the MIT Map-Reduce utility should create. For example, if the --np=100 option is used, only 100 compute tasks are created and each compute task will process a block of the total input data instead of a single data. The block size is determined by the MIT Map-Reduce utility.

With the --np option, one can handle a large number of input data files easily. However, this approach executes the application multiple times – as many as the number of input data files. There is a significant overhead cost associated with the repeated startup of the application, especially program environments such as MATLAB. One way to eliminate the overhead cost is to launch the application once and process all the data assigned to each compute task. This requires modifying the wrapper script shown in Fig. 4 in addition to modifying the job submission script and the run scripts shown in Figs. 6 and 7. This feature can be invoked by an example command as shown in Fig. 8. In this case, the command specifies a wrapper script, MatlabCmdMulti.sh, to handle the multiple lines of input and output file lists, created with the --apptype=mimo option. Also, with the --ext=gray option, the extention is renamed as ".gray" instead of the default, ".out".

```
$ LLMapReduce –-mapper MatlabCmdMulti.sh \
           –-input input –-output output \
           --np N --apptype mimo --ext gray
```

Fig. 8. An example Map-Reduce job to eliminate the overhead cost associated with the multiple execution of an application in the default processing model.

An example wrapper script, MatlabCmdMulti.sh, for the --apptype=mimo option is presented in Fig. 9. This script reads in the input and output file names from the generated file and provided through the run scripts, which are also generated by the MIT Map-Reduce utility. This script launches the application (MATLAB in this case) once and processes all the data, assigned by a dynamically generated file.

```
cat<<EOF|matlab -nodisplay -singleCompThread
inFile='$1'; fid=fopen(inFile);
tline=fgets(fid);
while ischar(tline)
  myStr=strsplit(tline);
  indata=deblank(myStr{1});
  outdata=deblank(myStr{2});
  imageConvert(indata, outdata);
  tline=fgets(fid);
end
fclose(fid);
EOF
```

Fig. 9. A wrapper script, MatlabCmdMulti.sh, to accept multiple lines of input and oout file names with the --apptype=mimo option.

With the --apptype=mimo option, the MIT Map-Reduce utility still generates a similar job submission script as shown in Fig. 6. However, it will generate different run scripts, named as run_llmap_x, which is shown in Fig. 10. The number x ranges from 1 to N, where N is the number of tasks defined by the --np option. In these run scripts, the wrapper script takes one input file, which is automatically generated by the MIT Map-Reduce utility. The input files named as input_x, that are also generated by the MIT Map-Reduce utility, have the list of input and output file names, one line per each input data file.

```
$ cat .MAPRED.2188/run_llmap_1 (for task 1)
#!/bin/bash
export PATH=${PATH}:.
MatlabCmdMulti.sh ./.MAPRED.2188/input_1
. . .

$ cat .MAPRED.2188/run_llmap_N (for task N)
#!/bin/bash
export PATH=${PATH}:.
MatlabCmdMulti.sh ./.MAPRED.2188/input_N
```

Fig. 10. A number of run scripts for all compute tasks generated by the MIT Map-Reduce utility.

B. *A Java Application*

One of the common Map-Reduce examples is a word frequency count application. In this example, a couple of java codes, WordFrequencyCmd.java and ReduceWordFrequency Cmd.java along with a few auxiliary codes, written by Fred Swartz [13] have been used. The WordFreqCmd.java code requires three command line inputs: input, output, and reference files. The reference file contains a list of words to be ignored for word counting. In order to comply with the MIT Map-Reduce API requirement, a wrapper script, WordFreqCmd.sh, has been created as shown in Fig. 11. In this wrapper script, the variables, $1 and $2, represent the input and output files, respectively.

```
#!/bin/bash
java WordFrequencyCmd $1 $2 textignore.txt
```

Fig. 11. A wrapper script for the WordFreqCmd.java code.

Also another wrapper script, ReduceWordFreqCmd.sh, is used to execute the ReduceWordFrequencyCmd.java code to collect the map process results as shown in Fig. 12. The reduce code scans the map results in the output directory and merges the results into a single file. The first argument ($1) for the reduce application is the location of the map process results and the second argument ($2) is the output name of the reduce application. Both arguments are provided by the MIT Map-Reduce utility.

```
#!/bin/bash
java ReduceWordFrequencyCmd $1 $2
```

Fig. 12. A wrapper script for the ReduceWordFreqCmd.java code.

With these two wrapper scripts, a Map Reduce job for the word frequency count can be launched by using the MIT Map-Reduce utility as shown in Fig. 13. In this example, the --distribution=cyclic option is used. With this option, the MIT Map-Reduce utility distributes the input data among the given number of compute tasks in a cyclic fashion. As mentioned in the previous MATLAB example, the Map-Reduce job launched by the command in Fig. 13 also incurs the computational overhead associated with the multiple startup of the word count application. The run script for the reduce task is submitted as a dependent job to the mapper job, which only uses one task currently. The java application, ReduceWordFrequencyCmd, scans the results in the given directory (output) and merges them into a single file (default name: llmapreduce.out).

```
$ LLMapReduce –-np 3
             –-mapper WordFreqCmd.sh \
             --reducer ReduceWordFreqCmd.sh \
             –-input input –-output output \
             --distribution cyclic
```

Fig. 13. A Map Reduce job for word frequency count using the MIT Map-Reduce utility.

In order to reduce the overhead cost, as was done in the MATLAB example, a new Map-Reduce job can be launched with the --apptype=mimo option as shown in Fig. 14. As mentioned previously, this option also requires the application modification so that it can read in multiple lines of input and output pairs provided by the MIT Map-Reduce utility.

```
$ LLMapReduce –-np 3
   –-mapper WordFreqCmdMulti.sh \
   --reducer ReduceWordFreqCmd.sh \
   –-input input –-output output \
   --apptype mimo
```

Fig. 14. A Map Reduce job for word frequency count using the MIT Map-Reduce utility with overhead cost reduction.

The WordFreqCmdMulti.sh script is shown in Fig. 15. It is similar to the script shown in Fig. 11 but executes a modified java application called WordFrequencyCmdMulti.

```
#!/bin/bash
java WordFrequencyCmdMulti $1 $2 \
                           textignore.txt
```

Fig. 15. A modified wrapper script for the WordFreqCmdMulti.java code.

The modified java code, WordFrequencyCmdMulti.java, has some additional lines, which reads in multiple lines of the input and output filename pairs. The input to the modified Java code is automatically generated by the MIT Map-Reduce utility. The original section of the code processes the given input file and writes the results to the given output file. As a result, the java code is invoked only once and processes all the input data assigned to its compute task.

IV. PERFORMANCE

In this section, we present the performance results of the --apptype=mimo (MIMO) option. We used the two example use cases from the previous section in addition to a user MATLAB application. Furthermore, we present the behavior of the three different options (DEFAULT, BLOCK, and MIMO) by varying the number of processes and the number of input data files.

The toy examples that we described previously are small in terms of number of input data files. The MATLAB application converts 6 images over 2 compute tasks. The Java application counts word frequency of 21 text files over 3 compute tasks. The Map-Reduce jobs were executed with the BLOCK and MIMO options, and the total processing time was measured. The speed up is calculated by the ratio between the time with the BLOCK option and the time with the MIMO option. The results are presented in Table 1. Although there are only a small number of data files assigned to each compute task in both cases, both examples show modest speed up with the MIMO option.

TABLE I. SPEED UP WITH TOY EXAMPLES

Example	**Type**	**Speed up**
Matlab	Multiple app launches (BLOCK)	1
	Single app launch (MIMO)	2.41
Java	Multiple app launches (BLOCK)	1
	Single app launch (MIMO)	2.85

A performance study with a user MATLAB application has been performed and the results are presented in Table 2. The MATLAB application does image processing, and the image files were distributed to 256 compute tasks. The number of input data files was 43,580 in this example. As MATLAB takes relatively significant time to launch as compared to other programs, the performance difference was significant. By using the MIMO option, the Map-Reduce job was able to run almost 12 times faster than that of the BLOCK option when comparing the job elapsed times between the two runs.

TABLE II. SPEED UP WITH A REAL WORLD APPLICATION

Example	**Type**	**Speed up**
Matlab	Multiple app launches (BLOCK)	1
	Single app launch (MIMO)	11.57

For the scalability study, we used the three different MIT Map-Reduce options (DEFAULT, BLOCK, and MIMO) with a MATLAB code that reads in a list of square matrix sizes and does multiplications of the given matrices. First, we created 512 input data files. For the scalability study, we run the simulation with various number of compute processes, ranging from 1, 2, 4, 8, 16, 32, 64, 128, and 256 for three different options. The results are presented in Figs. 16 and 17.

Fig. 16 shows the computational overhead associated with the cost of multiple application start-ups when dealing with more than one input data file per compute task. While the cases for the DEFAULT and BLOCK options show that the average overhead cost per compute process decreases linearly as the number of compute processes is increased, the overhead cost for the MIMO option remains relatively flat. As far as the overhead cost is concerned, both DEFAULT and BLOCK options show similar overhead, although the BLOCK option shows slightly smaller cost. The MIMO overhead cost is significantly smaller than those of the other two options. Thus the gap in the overhead cost between the MIMO and the other two options becomes significant especially when each compute task processes a large number of data files. Fig. 16 clearly shows the benefits of the MIMO option when dealing with a large number of input data files per compute task.

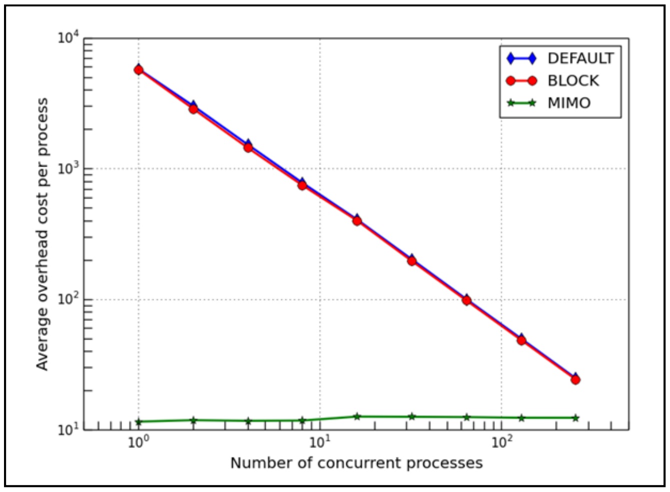

Fig. 16. The computational overhead cost when varying the number of compute processes, which changes the number of the input data files per compute task.

Fig. 17 shows the speed-up based on job elapsed times for the three different options with varying number of compute processes. The speed-up is calculated by the ratio between the DEFAULT job elapsed time obtained with one compute process and the other job elapsed times. Throughout all the numbers of the concurrent processes, the MIMO option performed the best, consitently outperforming the other two options. The BLOCK option performed slightly better than the DEFAULT option but the difference is marginal. As the number of concurrent processes is increased and because the overhead cost per compute task is diminishing, the gap between the speed up of MIMO job and the other two results gets closer. If each compute task processes only one data, the results of all three options will converge at the same point.

V. SUMMARY

MIT Map-Reduce utility has been developed and deployed on the MIT SuperCloud to support scientists and engineers at MIT Lincoln Laboratory. With the MIT Map-Reduce utility, users can deploy their MapReduce-style applications quickly on to the MIT SuperCloud infrastructure. The MIT Map-Reduce utility can work with any executable application without the need for any modifications. However, for improved performance, the MIT Map-Reduce utility provides an option to consolidate multiple input data files per compute task as a single stream of input with minimal changes to the target application. This enables users to cut down the computational overhead associated with the cost of repeated application start-ups when dealing with more than one input data file per compute task. With a small change in a sample MATLAB image processing application, we have observed approximately 12x speed up by reducing the overhead associated with the repeated application start-ups. Currently the MIT Map-Reduce utility can work with handful of schedulers including SLURM, Grid Engine and LSF.

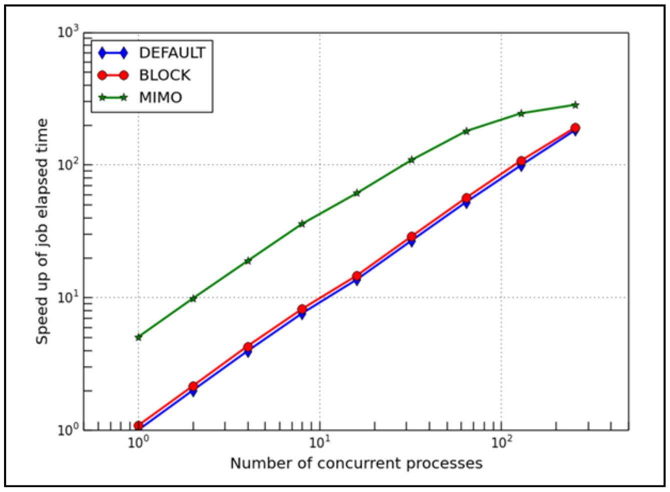

Fig. 17. The speed-up of job elapsed times with respect to the default job elapsed time with one compute process when varying the number of compute processes, which in turn changes the number of the input data files per compute task.

REFERENCES

[1] A. Reuther, J. Kepner, W. Arcand, D. Bestor, W. Bergeron, C. Byun, M. Hubbell, P. Michaleas, J. Mullen, A. Prout, & A. Rosa, “LLSuperCloud: Sharing HPC Systems for Diverse Rapid Prototyping,” IEEE HPEC, Sep 10-12, 2013, Waltham, MA.

[2] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Proceedings of the 2004 OSDI Conference, December 5, 2004, San Francisco, CA.

[3] Apache Hadoop (http://hadoop.apache.org/).

[4] Apache Hadoop 1.2.1 Documentation: Hadoop Streaming (http://hadoop.apache.org/docs/r1.2.1/streaming.html).

[5] C. Byun, W. Arcand, D. Bestor, W. Bergeron, M. Hubbell, J. Kepner, A. McCabe, P. Michaleas, J. Mullen, D. O’Gwynn, A. Prout, A. Reuther, A. Rosa, C. Yee, “Driving Big Data With Big Compute,” IEEE HPEC, Sep 10-12, 2012, Waltham, MA.

[6] Apache Hadoop 1.2.1 Documentation: HDFS (http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html).

[7] Slurm workload Manager (http://slurm.schedmd.com/slurm.html).

[8] Univa Grid Engine Software (http://www.univa.com/products/grid-engine.php).

[9] Open Grid Scheduler/Grid Engine (http://gridscheduler.sourceforge.net/).

[10] The Son of Grid Engine project (https://arc.liv.ac.uk/trac/SGE).

[11] IBM Platform LSF (http://www-03.ibm.com/systems/platformcomputing/products/lsf/).

[12] The OpenSFS and Lustre Community Portal (http://lustre.opensfs.org/).

[13] F. Swartz, a word frequency count example written in Java (https://code.google.com/p/nealsproject/source/browse/FHXExtraction/src/wordcomparison/?r=38).