Cloud Computing Homework 2

Problem 1

1. ***Speedup***:

   - **Definition**: Speedup of the cloud system a (with latency L_a) compared to cloud system b (with latency L_b) in latency is defined by the following formula:

     $$S_{(a,b)} = \frac{L_b}{L_a}$$

   - **Physical meaning**: Speedup is a number that measures the relative performance of two cloud systems processing the same (fix-sized) problem. More technically, it is the improvement in speed of execution of a task executed on two cloud architectures with different resources.

   - **Target scenarios**: Measuring the improvement of the speed of processing a fixed amount of data as the cloud architecture is improved.

2. ***Efficiency***:

   - **Definition**: The cloud efficiency E_a of cluster a containing k types of machines is defined as follows:

     $$E_A = \frac{S_{(a,u)}}{N_a}$$

     Where $S_{(a,u)}$ denotes the speedup of a in comparison to u. u denotes a 1-ECU instance and N_a is defined as:

     $$N_a = \sum_{i=1}^{k} n_i \times c_i$$

     With n_i denoting the number of machines for type-i (which also reflects the maximum speedup that can be achieved theoretically) and c_i denoting the computing power of each machine instance of type-i.

   - **Physical meaning**: N_a essentially gives the theoretical maximum speedup that can be achieved by cluster a compared to the "unit" cluster u. Hence, the efficiency metric measures in reality the percentage of the theoretical peak performance that can be achieved.

   - **Target scenarios**: The measurement of cloud efficiency is helpful in identifing cloud wastes. It helps measure the gap between the amount of services the user's buying and the amount that is actually being used, so that the user can see how much cloud resources purchased are wasted (unused) and hence think of better strategies to utilize them.

3. ***Productivity***:

   - **Definition**: The cloud productivity of a cloud configuration a is defined as follows:

$$P(a) = \frac{p(a) \times w(a)}{C(a)}$$

Where $p(a)$ is a performance metric used, which, for example, could be speed or throughput. $w(a)$ is the QoS of the cloud, which can be approximated by the service availability. $C(a)$ represents the cost required for the user to rent resources in forming the cluster a.

- **Physical meaning**: The productivity essentially shows the performance/price ratio of the cloud system a (that is, the available performance that can be obtained by spending per unit cost). Larger productivity means that the cloud system is more cost-effective, and the lower productivity means that the system is less economical.

- **Target scenarios**: By measuring the productivity the cloud providers can understand how much are costs when a desired service is provided to the users so that they can control and manage their costs more efficiently. In addition, since users might worry if they spend higher than they expected (higher than that the services worth) on purchasing the cloud services, productivity can be a suitable criteria for the users to judge the services' cost-efficiency.

4. ***System Availablity***:

- **Definition**: The availability A of the cloud system b is defined as follows:

$$A_b = \frac{MTTF}{MTTF + MTTR}$$

Where MTTF denotes Mean Time To Failure, the average time taken for the cloud system to go from up to down. MTTR denotes Mean Time To Repair, the average time taken for the cloud system to go from down to up.

- **Physical meaning**: System availability determines the instantaneous performance of the cloud system at any given time based on time duration between its failure and recovery.
- **Target scenarios**: Cloud availability is related to cloud reliability. By checking the system availability the cloud users can know the extent to which the cloud products and services purchased are accessible anytime and anywhere, or functioning reliably and as expected.

5. ***Scalability***:

- **Definition**: The cloud scalability of scaling from cloud system a_1 to cloud system a_2 is defined as follows:

$$S_{(a_1,a_2)} = \frac{P(a_2)}{P(a_1)} = \frac{p(a_2) \times w(a_2) \times C(a_1)}{p(a_1) \times w(a_1) \times C(a_2)}$$

Where $P(a_2)$ and $P(a_1)$ are the productivities calculated by the formula mentioned in 3.

- **Physical meaning**: Scalability evaluates the economy of scale by a pair of productivity ratios. The higher the value of a scalability measure, the more opportunity exists to target the desired scaling scheme.
- **Target scenarios**: The scalability measurement based on productivities is important for the business companies to determine the economically optimal scaling strategy. When scaling the cloud services in response for the user demands, such criteria helps determines whether or not such scaling would be worthwhile (in costs and benefits).

6. ***Return On Investment (ROI)***

   - **Definition**: The Return On Investment (ROI) is defined as follows:

     $$ROI = \frac{G_a - C_a}{G_a}$$

     Where G_a denotes the earnings as a result of the investment of the cloud architecture a. C_a denotes the costs of investing in the cloud architecture a.
   - **Physical meaning**: ROI indicates whether or not and to what extent the cloud system can earn profits. A positive ROI indicates that the projected gains compare favorably to the projected investment costs, whereas a negative one indicates that the projected costs is more favorable than the projected gains.
   - **Target scenarios**: ROI can be used as a hard figure to justify the investments for managing costs and profits. For example, many organizations that choose the cloud migration path tend to use an ROI assessment to determine if a cloud migration will save the company money, help them achieve their organizational goals, and how long it will take them to recoup that investment.

Problem 2

1. ***The architecture of Apache HDFS***

   HDFS has a master/slave architecture containing a single NameNode (the master), a secondary NameNode and many DataNodes (the slaves), mounted directly with a Filesystem in Userspace (FUSE).

   - **NameNode**: The single NameNode serves as a master managing the mapping of the blocks to DataNodes. It also manages the file system's metadata and namespace, like all information regarding the location of input splits/blocks in all DataNodes
   - **DataNode**: The DataNode server manages the storage attached to the node. Each DataNode is responsible for the storing and retrieving of its file blocks. Each DataNode serves up blocks of data over the network using a block protocol specific to HDFS. The underlying file system uses TCP/I = sockets for communication. Clients use remote procedure call (RPC) to communicate between each other. When storing large files, the DataNodes achieve reliability by communicating and replicating the data across multiple hosts.
   - **Secondary NameNode**: The secondary NameNode regularly connects with the primary NameNode and builds snapshots of the primary NameNode's directory information, which the system then saves to local or remote directories. Such images can be used to restart a failed primary NameNode without having to replay the entire journal of file system actions and then edit the log to create an up-to-date directory structure.
   - **Filesystem in Userspace (FUSE)**: HDFS is mounted directly with the Filesystem in Userspace (FUSE), a virtual file system on Linux and some other Unix systems that facilitates file accessesthrough native APIs, the command-line interface, web application (web app) over HTTP, or via third-party network client libraries.

2. ***How HDFS stores an 812MB piece of data in Hadoop 2.0***

   First the 812MB piece of data would be broken into blocks (since each block has default size 128MB, the data would by default be broken into 7 blocks, 6 full and 1 with 44MB). Then such blocks are distributed to the DataNodes and the information in the NameNode would be updated. Then the data

streamer would ask the NameNode for the suitables DataNodes for storing replicas of the blocks and replicates the blocks one by one according to the replication factor (which by default is 3, meaning that each block would have 3 extra copies in different DataNodes) for availability.

3. ***How Hadoop 2.0 manages job scheduling***

   By default, Hadoop uses first-in-first-out (FIFO) scheduling, and optionally five priority heuristics to schedule jobs from a work queue. The job scheduler was refactored out of the JobTracker, while adding the ability to use an alternate scheduler. Two scheduling policies offer the alternative following choices:

   - **Fair Scheduler**: It was developed by Facebook. The goal of the Fair Scheduler is to provide fast response times for small jobs and quality of service (QoS) for production jobs. It has three basic concepts: (1) jobs are grouped into pools; (2) each pool is assigned a guaranteed minimum share; and (3) excess capacity is split between jobs. By default, jobs that are uncategorized go into a default pool. Pools have to specify the minimum number of map slots, reduce slots, and a limit on the number of running jobs.
   - **Capacity Scheduler**: Developed by Yahoo!, the Capacity Scheduler supports several features that are similar to the Fair Scheduler. Queues are allocated a fraction of the total resource capacity. Free resources are allocated to queues beyond their total capacity. Within a queue a job with a high level of priority has access to the queue's resources. There is no preemption once a job is running with Hadoop.

4. ***Improvement over the earlier version of HDFS***

   - **HDFS federation**: Hadoop 2.0 feature HDFS Federation allows horizontal scaling for Hadoop distributed file system (HDFS). This is one of the many sought after features by enterprise class Hadoop users such as Amazon and eBay. HDFS Federation supports multiple NameNodes and namespaces.
   - **NameNode High Availability**: Hadoop 2.0 Architecture supports multiple NameNodes to remove this bottleneck. Hadoop 2.0, NameNode High Availability feature comes with support for a Passive Standby NameNode. These Active-Passive NameNodes are configured for automatic failover.
   - **YARN's adaptability**: Hadoop 2.0 provides YARN API's to write other frameworks to run on top of HDFS. This enables running Non-MapReduce Big Data Applications on Hadoop. Spark, MPI, Giraph, and HAMA are few of the applications written or ported to run within YARN.
   - **Separating JobTracker's functions**: YARN's The Next Generation MapReduce framework (MRv2) divides the two major functions of the JobTracker, resource management and job scheduling/monitoring, into separate components: The new ResourceManager managing the global assignment of compute resources to applications and the per-application ApplicationMaster managing the application's scheduling and coordination. Such division provices better resource management services and hence improves cluster efficiency.

Problem 3

a. **AIRS Cloud architecture**: AIRS core architecture involves 5 major resource clusters:

- **Infrastructure as a Service (IaaS)** include virtual machines, block storage, file storage, object storage, virtual switching networks, virtual routes, virtual security groups, etc. The resource adopts OpenStack+KVM+Ceph.

- **Platform as a Service (PaaS)** include application orchestration, application middleware (Webpress, Nginx, MySQL, etc.), DevOps center, micro service governance center, etc. The resource pool mainly adopts the software stack of Kubernetes+Docker.
- **ICT edge platform services** include edge computing resources such as 5G core network, 5G network management, service center, API center and CT service center. The resource pool mainly adopts the software stack of Kubernetes+Docker+KVM and the 3GPP edge framework.
- **Hadoop data platform service (BDaaS, Big Data as a Service)** for big data. These services include HDFS Distributed Big Data File System, YARN Scheduling System, MapReduce Distributed Framework and Spark Memory Computing Framework. The resource pool mainly adopts the software stack of Hadoop ecosystem.
- **AI platform service (AIaaS, AI as a Service)** for deep learning artificial intelligence. These services include, but are not limited to, training framework, general data set, reasoning framework, AutoML and other resources. The resource pool mainly adopts the software stack of Kubernetes+Docker.

b.

- I would use the **PaaS Cloud Resource Platform** since it contains powerful application operation and deployment platforms including application orchestration (so that I can divide the application into sub-modules and deploy them in different containers), application middleware (Webpress, Nginx, MySQL, etc. Those middlewares make deployment easier), DevOps center (facilitating my operation), micro service governance center (making the division of labor of my application more clear), etc.
- I would use **container** to deploy my application since the docker container supports multiple IP addresses to be attached and that it's relatively light-weighted and easier to deploy. Together I would use middlewares like Nginx and MySQL to do load balancing and facilitate my application functions. I would use DevOps for operations.

c.

- I would use the **BDaaS Cloud Resource Platform** since it adopts the software stack of the distributed Hadoop ecosystem, including HDFS Distributed Big Data File System (convenient for managing the distributed file system and the distributed input/output stream), YARN Scheduling System (powerful in job scheduling and resource management), MapReduce Distributed Framework, Spark Memory Computing Framework (facilitating distributed computation) and data acquisition tools like Sqoop, Flume and Kafka (convenient for obtaining data from other edge nodes or the database). I would prepare the dataset and run the program in **Hadoop** on the platform.

Problem 4

Environment:

The following experiments are conducted on the Hadoop cluster in the AIRS cloud. I selected "huge" as the suitable data size for all the experiments since the running time is long and convenient enough to help characterize speedups and scalability. The datasets for different applications are prepared using Java codes:

For **pagerank**, the file is in /home/accountX/HiBench3/autogen/target/autogen-8.0-SNAPSHOT-jar-with-dependencies.jar:

```
    jar /home/accountX/HiBench3/autogen/target/autogen-8.0-SNAPSHOT-jar-with-
dependencies.jar HiBench.DataGen -t $APP_TYPE$ -b
hdfs://cuhkcluster/dataspace/teamY/HiBench/$APP_TYPE$ -n Input -m $MAPPER_NUMBER$
-r $REDUCER_NUMBER$ -p $DATA_SIZE$ -pbalance -pbalance -o text
```

For **TeraGen**, the file is in /usr/hdp/3.0.1.0-187/hadoop/../hadoop-mapreduce/hadoop-mapreduce-examples.jar:

```
    jar /usr/hdp/3.0.1.0-187/hadoop/../hadoop-mapreduce/hadoop-mapreduce-
examples.jar teragen -D mapreduce.job.maps=$MAPPER_NUMBER$ -D
mapreduce.job.reduces=$REDUCER_NUMBER$ -D mapreduce.job.queuename=teamY
$DATA_SIZE$ hdfs://cuhkcluster/dataspace/teamY/HiBench/Terasort/Input
```

For **WordCount**, the file is in /usr/hdp/3.0.1.0-187/hadoop/../hadoop-mapreduce/hadoop-mapreduce-examples.jar:

```
    jar /usr/hdp/3.0.1.0-187/hadoop/../hadoop-mapreduce/hadoop-mapreduce-
examples.jar randomtextwriter -D mapreduce.randomtextwriter.totalbytes=... -D
mapreduce.randomtextwriter.bytespermap=... -D mapreduce.job.maps=$MAPPER_NUMBER$ -
D mapreduce.job.queuename=teamY -D mapreduce.job.reduces=$REDUCER_NUMBER$
hdfs://cuhkcluster/dataspace/teamY/HiBench/Wordcount/Input
```

The definitions of "**huge**" data size for different applications are listed below:

For **TeraSort**:

```
hibench.terasort.huge.datasize      320000000
```

For **WordCount**:

```
hibench.wordcount.huge.datasize     24000000000
```

For **PageRank**:

```
hibench.pagerank.huge.pages          2000000
hibench.pagerank.huge.num_iterations     3
hibench.pagerank.huge.block          0
hibench.pagerank.huge.block_width        16
```

Definitions:

1. The speedup (of upgrading from cluster A to cluster B) $S_u(A,B)$ is defined as:

$$S_u(A,B) = \frac{T_A}{T_B}$$

   Where T_A and T_B represents the execution time of a certain task on cluster A and cluster B, respectively. Such speedup can be used to quantify how much the performance of cluster A on certain task is better than that of cluster B.

2. The scalability (of upgrading from cluster A to cluster B) $S(A,B)$ is defined as:

$$S(A,B) = \begin{cases} \frac{S_u(A,B)-1}{R(A,B)-1} & A \neq B \\ 0 & A = B \end{cases}$$

   Where $S_u(A,B)$ is the speedup of upgraing from cluster A to cluster B. $R(A,B)$ is expected to be the ratio of the number of nodes participating scaling from cluster A to B, which reflects the number of nodes being scaled and would be defined differently in different experiment sets. Such criteria characterizes the scaling ability of a cluster: A positive scalability indicates that the scaling of the cluster can improve the speed. A value greater than or equal to 1 indicates that such scaling is worthy in that its actual performance improvement is compatible to or even exceeds the improvement indicated by the number of scaling nodes. A value less than or equal to 0 indicates that such scaling brings no improvement or even negative effects on cluster performance.

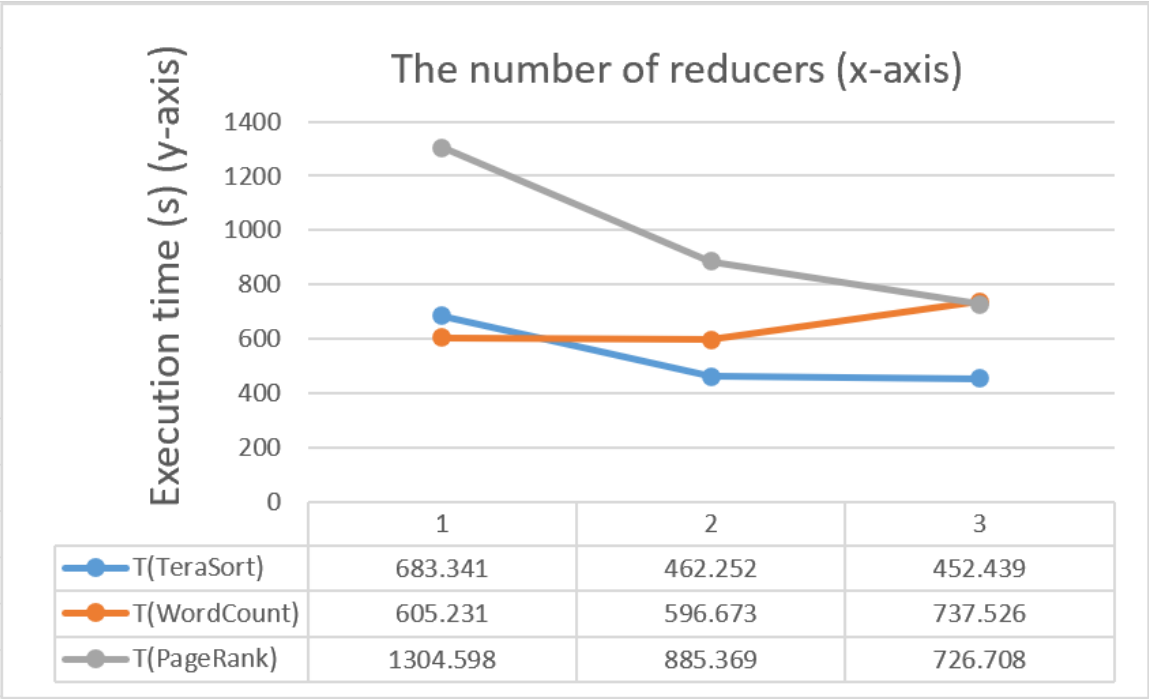
Experiment set A: Fixing #mappers and increasing #reducers

a. Experimental process:

In this experiment I aim to examine the scalability of the reducers in Hadoop cluster on AIRS cloud. The number of mapper is fixed to 1 and the number of reducers varies from 1 to 3. I run TeraSort, WordCount and PageRank in HiBench on clusters (with different number of reducers) in order to characterize the scalability of the Hadoop cluster on different applications, which can be affected due to the differences in algorithms. After running those applications, the running time and throughput information are collected from file *hibench.report* and the information regarding I/O and CPU execution time are collected from *bench.log* under the report folder.

b. Results:

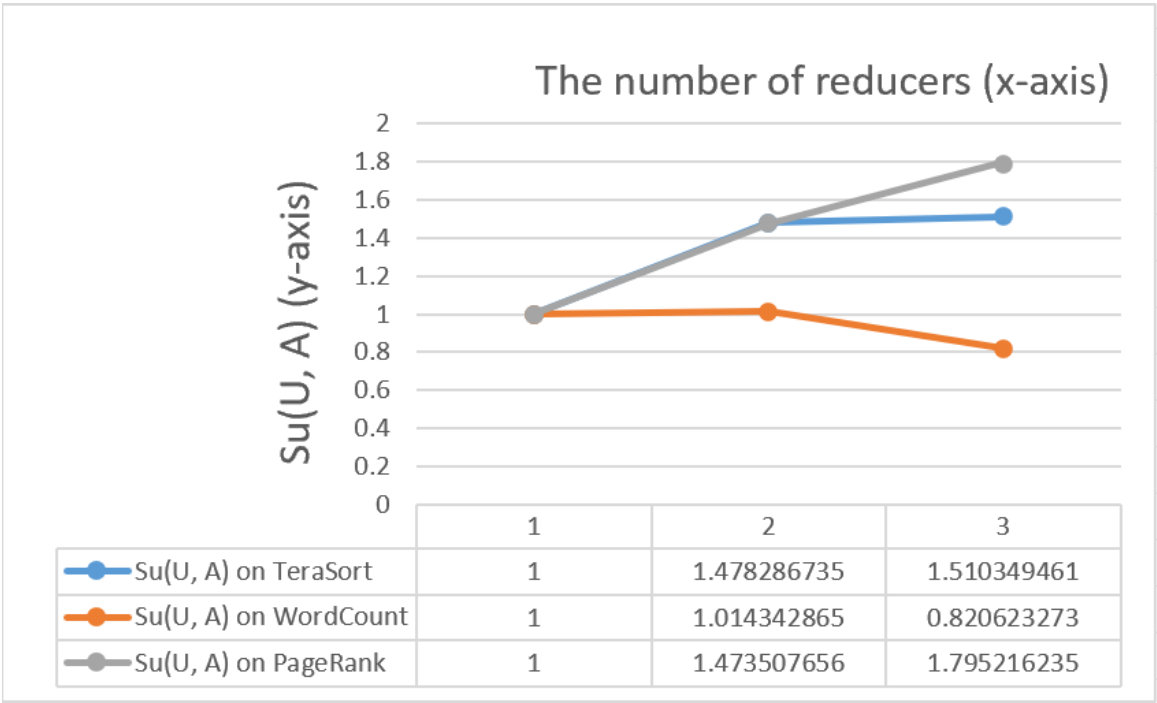
Figure 1 shows how the execution time T of a certain application changes with respect to the number of reducers.



	1	2	3
T(TeraSort)	683.341	462.252	452.439
T(WordCount)	605.231	596.673	737.526
T(PageRank)	1304.598	885.369	726.708

Figure 1: The change of execution time with respect to # reducers

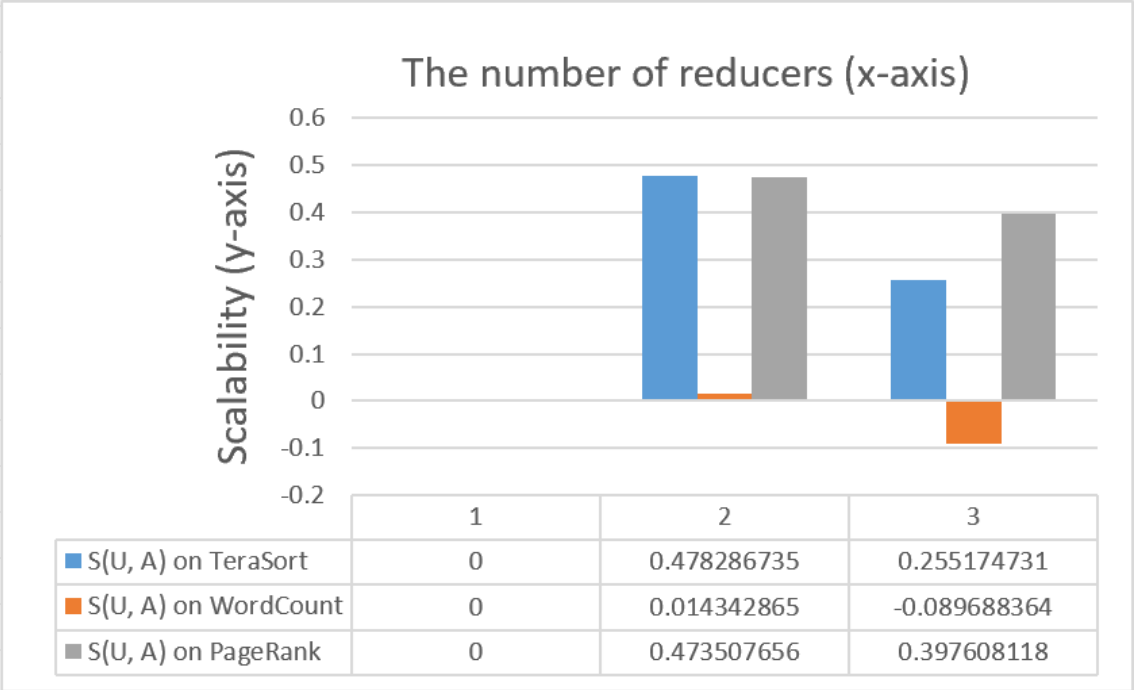
Figure 2 shows how the speedup of the particular applications ($Su(U, A)$) change with respect to the number of reducers used (where A denotes the current cluster with certain number of reducers being examined and U denotes the cluster where 1 mapper and 1 reducer are used and other settings remain the same as cluster A).



	1	2	3
Su(U, A) on TeraSort	1	1.478286735	1.510349461
Su(U, A) on WordCount	1	1.014342865	0.820623273
Su(U, A) on PageRank	1	1.473507656	1.795216235

Figure 2: The change of speedup with respect to # reducers

Figure 3 shows how the scalability of a cluster on a certain task changes with respect to the number of reducers used in that cluster.



	1	2	3
S(U, A) on TeraSort	0	0.478286735	0.255174731
S(U, A) on WordCount	0	0.014342865	-0.089688364
S(U, A) on PageRank	0	0.473507656	0.397608118

Figure 3: The change of scalability with respect to # reducers

c. Discussions:

For **TeraSort**, the execution time decreases as the number of reducers increase, meaning that increasing the reducer number accelerates completing the sorting task. As shown from figure 4, the processor load per server is significantly decreased as the number of reducers increase, indicating that more reducers help balance the processing loads of the servers.

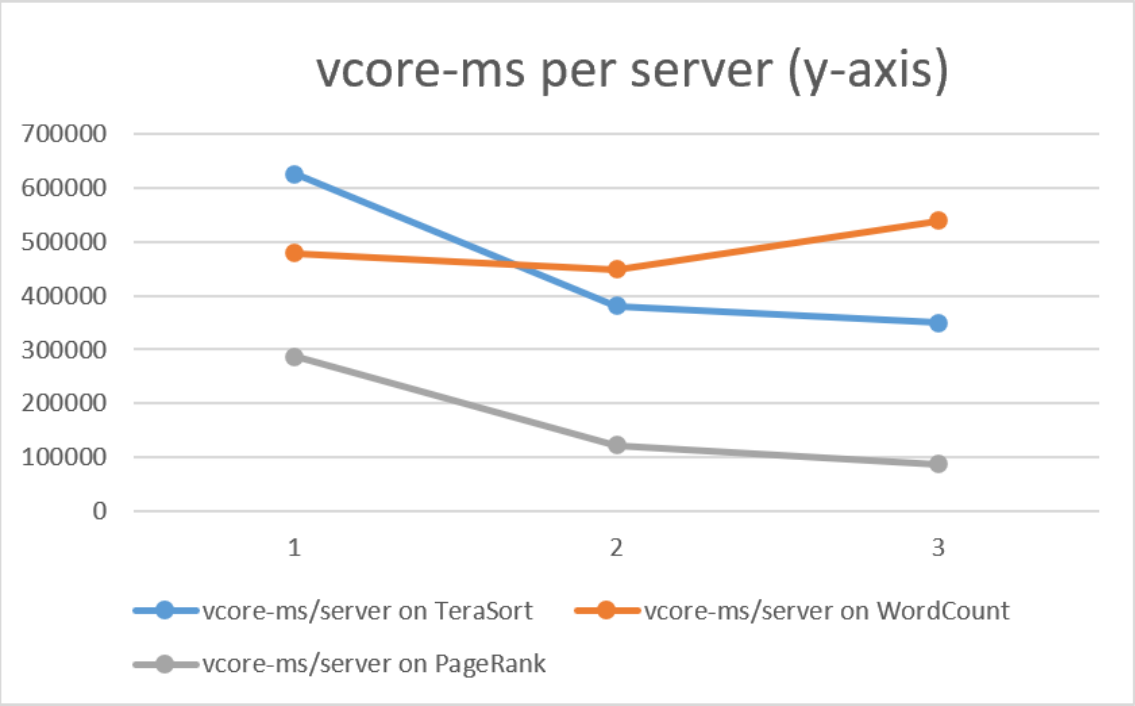


Figure 4: vcore execution time per server with respect to different number of reducers

However, we may also observe that the speedups didn't reach the ideal values (which, for reducer number=2 should be 2 and for reducer number=3 should be 3 in correspondance to the number of nodes scaled). Silimar situation occured in scalability (where the ideal values should be 1 but getting less than 1 instead). One potential reason is that the unavoidable bottleneck of the map tasks' execution time limits the performance of the cluster. Furthermore, IO time plays an increasingly important role as the number of reducers increase: As figure 5 indicates, the number of map outputs that requires shuffling increase as more reducers are required, which increases the amount of time required to partition and schedule those outputs and hence demands more overheads (such as metadata on which reducer the merged map outputs should be distributed to, as well as the time taken to schedule those map outputs to more reducers).

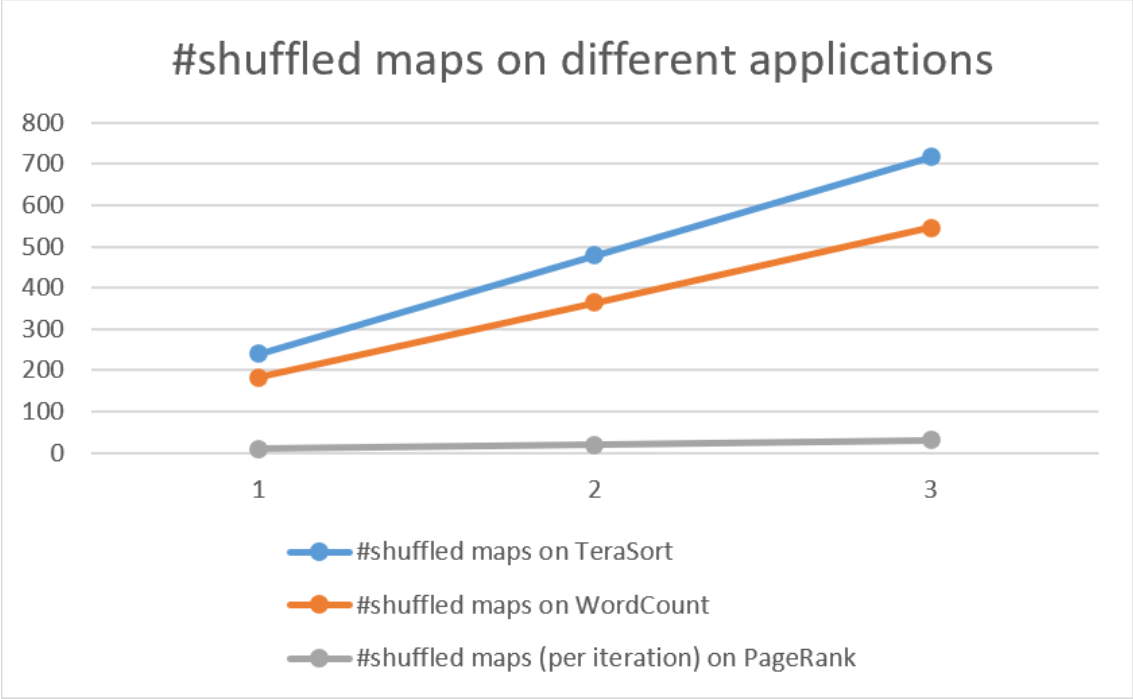


Figure 5: The change of # shuffled maps with respect to # reducers

For **PageRank**, the execution time decreases significantly as the number of reducers used increase, and the speedup also increases. Such phenomenon indicates that when running computationally intensive jobs like PageRank (requiring more computations done on iterative computations of page ranks among many web pages) on a given upgraded cluster, the speedup that can be achieved is stronger than that when running computationally non-intensive jobs since the running time on CPUs plays an important part in the optimization of general performance (as indicated in figure 4).

Still, we observe the fact that the speedup and scalability didn't reach the ideal values (for speedup, 2 or 3 and for scalability, 1). This is partly due to the IO costs and scheduling overheads taken in the shuffle stage, where mapped partial page rank product results are scheduled and distributed to different reducers. Also, reconnecting to the cluster for iterative page rank computations costs time (given in the environment that the maximum number of iterations is set to 3), which can't be optimized by simply scaling.

For **WordCount**, the execution time, the speedup, as well as the scalability remains stable as the number of reducers increase. In the case where the number of reducer is 3 even negative scalability occured. The potential reason is that the WordCount algorithm requires more efforts on mapping, characterized by "spliting" the sentences into multiple words, than on reducing, characterized by just adding the numbers given in the value part. As figure 6 shows, the time taken by the map tasks is significantly greater than that taken by the reduce tasks, making the effect of scaling the number of reducers on execution time less significant. On the other hand, the shuffling, scheduling and communication costs, as indicated in figure 5, is also increased due the increasing number of mapped outputs.

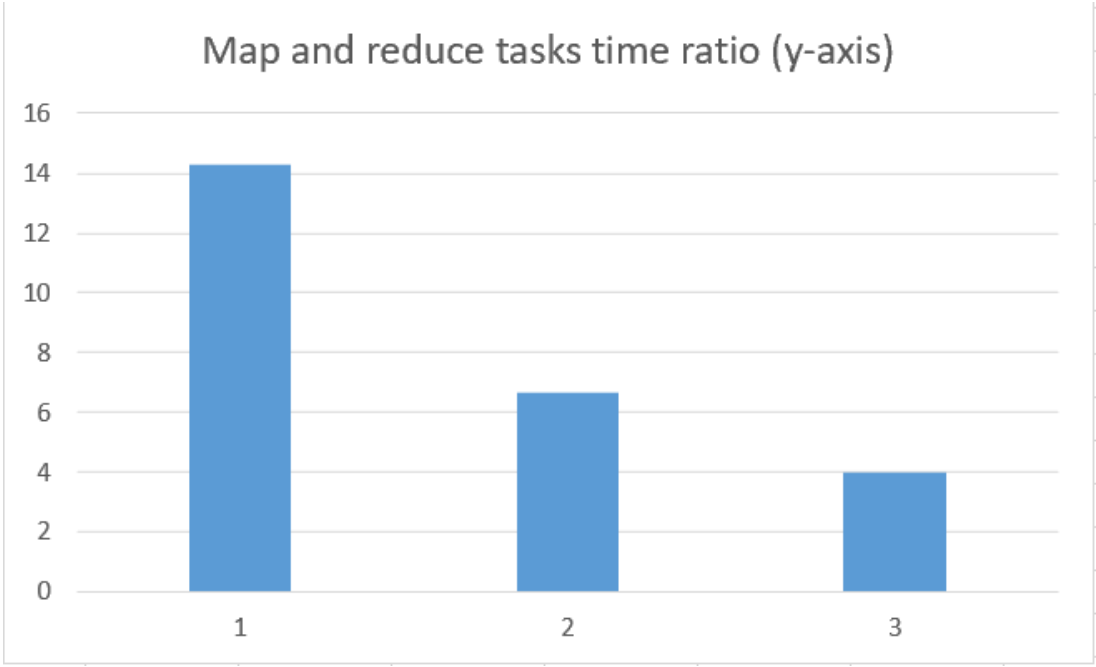


Figure 6: The change of Map/Reduce time ratio with respect to # reducers

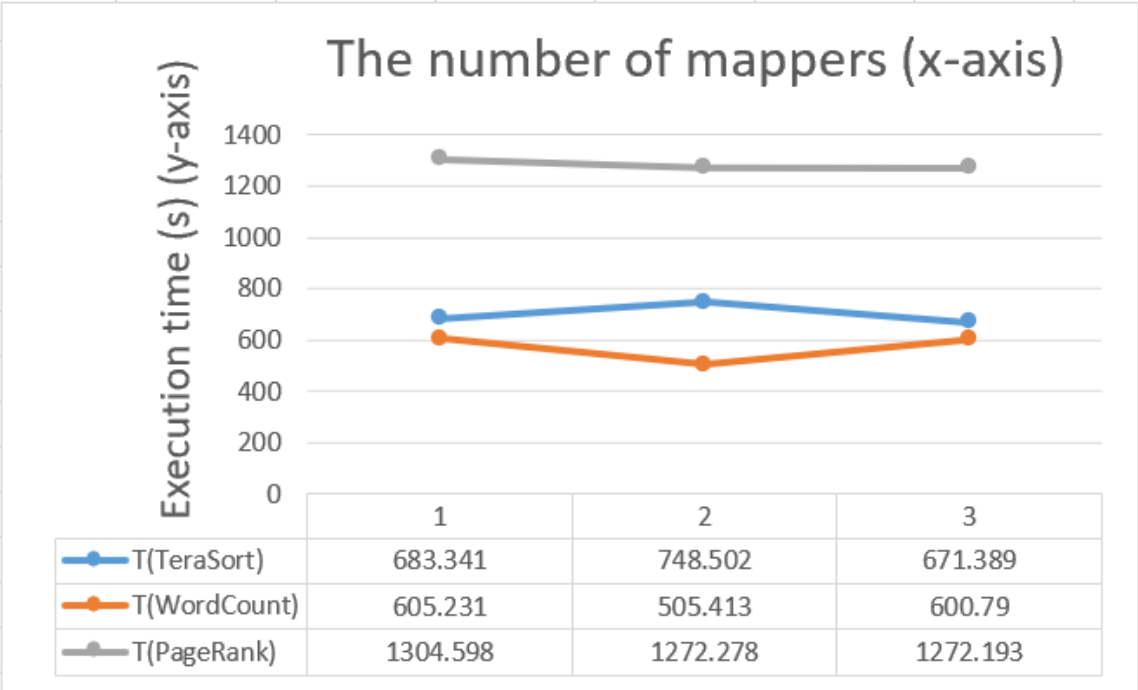
Experiment set B: Fixing #reducers and increasing #mappers

a. Experimental process:

In this experiment I aim to examine the scalability of the mappers. The number of reducer is fixed to 1 and the number of mappers varies from 1 to 3. I run TeraSort, WordCount and PageRank in HiBench on clusters (with different number of mappers) in order to characterize the scalability on different applications. After running those applications, the running time and throughput information are collected from file *hibench.report* and the information regarding I/O and CPU execution time are collected from *bench.log* under the report folder.

b. Results:

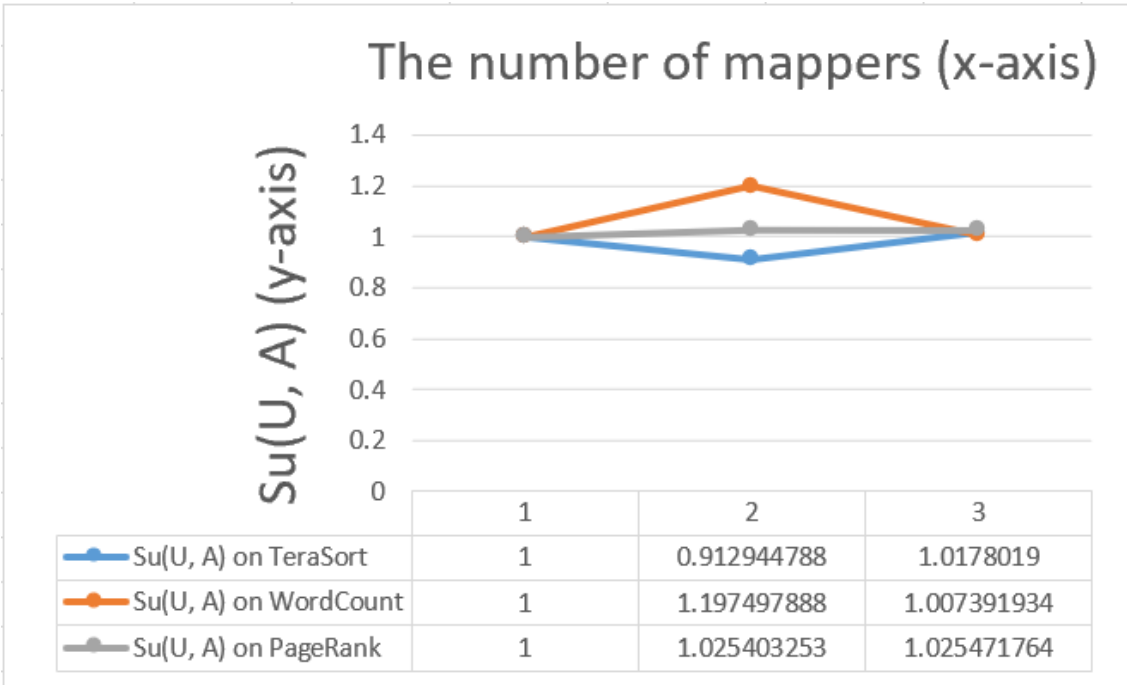
Figure 7 shows how the execution time T of a certain application changes with respect to the number of mappers.



	1	2	3
T(TeraSort)	683.341	748.502	671.389
T(WordCount)	605.231	505.413	600.79
T(PageRank)	1304.598	1272.278	1272.193

Figure 7: The execution time changing with respect to # mappers

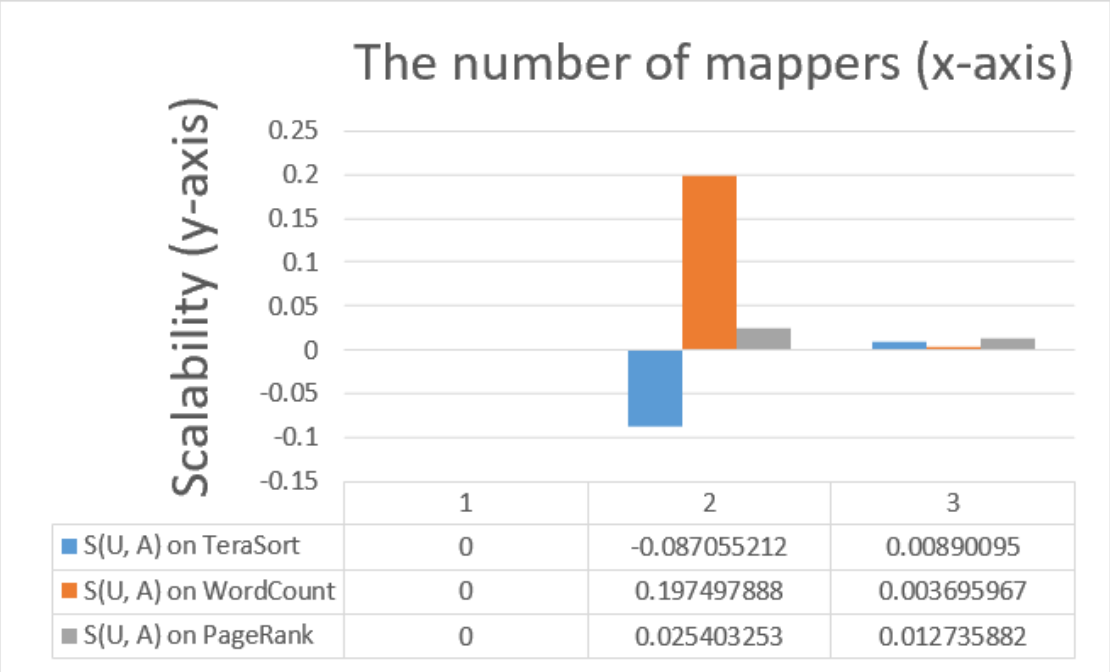
Figure 8 shows how the speedup of the particular applications ($Su(U, A)$) change with respect to the number of mappers used (where A denotes the current cluster with certain number of reducers being examined and U denotes the cluster where 1 mapper and 1 reducer are used and other settings remain the same as cluster A).



	1	2	3
Su(U, A) on TeraSort	1	0.912944788	1.0178019
Su(U, A) on WordCount	1	1.197497888	1.007391934
Su(U, A) on PageRank	1	1.025403253	1.025471764

Figure 8: The speedup changing with respect to # mappers

Figure 9 shows how the scalability of a cluster on a certain task changes with respect to the number of mappers used in that cluster.

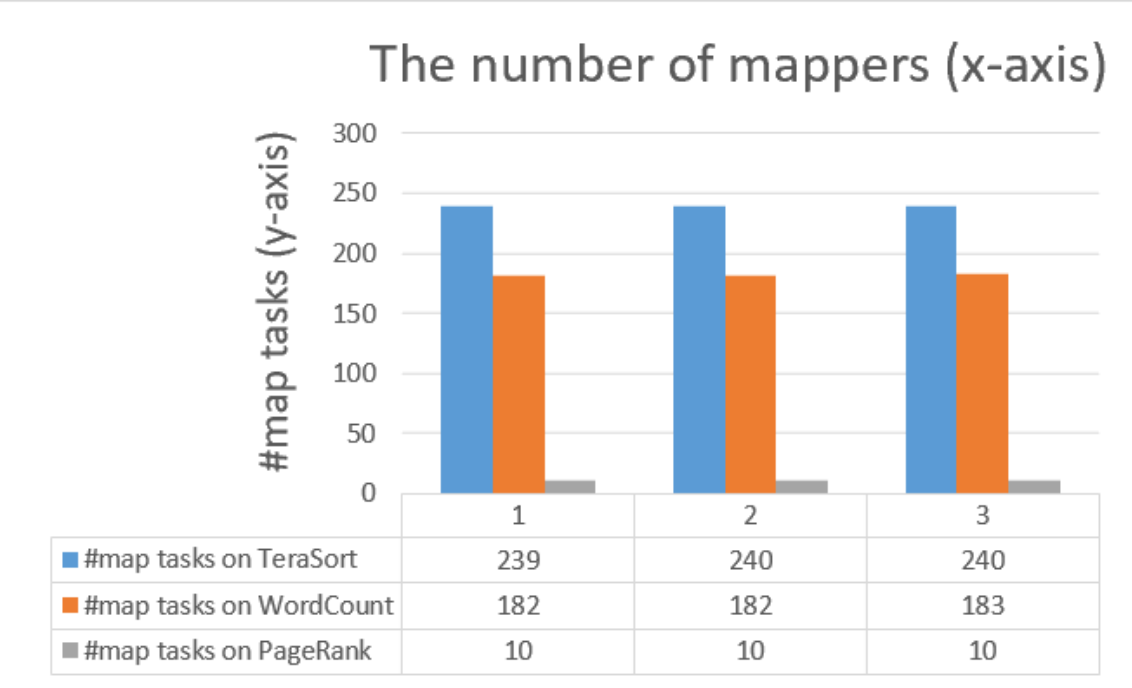


	1	2	3
S(U, A) on TeraSort	0	-0.087055212	0.00890095
S(U, A) on WordCount	0	0.197497888	0.003695967
S(U, A) on PageRank	0	0.025403253	0.012735882

Figure 9: The change of scalability with respect to # mappers

c. Discussions:

For all the applications, the execution time, the speedup and the scalability remain stable as the number of mapper varies from 1 to 3. The reason is that after a task had been submitted to Hadoop, the actual number of map tasks is limited by both the number of mappers set by the user and the HDFS disk block size. Since each map task can only deal with one map block, the number of map tasks essentially depends on data size, which determines the number of blocks to be allocated given a fixed disk block size. It is also shown in figure 10 that the number of map tasks for different applications remain the same as the data size remains consistent.



	1	2	3
#map tasks on TeraSort	239	240	240
#map tasks on WordCount	182	182	183
#map tasks on PageRank	10	10	10

Figure 10: The change of # map tasks with respect to # mappers

Problem 5

In this experiment I aim to measure the AIRS cloud's ability in 5 aspects: **Speedup**, **Scalability**, **QoS** and **Bandwidth** using matrix multiplication as an application benchmark. The **speedup** and **scalability**, which is defined the same as that defined in problem 4, characterize the AIRS cloud's ability to scale-up or scale-out. The **QoS** characterizes the availability of the AIRS cloud, the ability that the AIRS cloud can remain up when running an application. The **Bandwidth** characterizes the I/O handling ability. It can be measured using the number of bytes written to HDFS divided by the time taken from the beginning of the completiong of reducing task to the completion of the entire map-reduce job (since the bytes written to HDFS must be the reducers' outputs to files, which takes place after the completion of reducing and before the completion of the job). Then I draw a **radar chart** and compare the AIRS cloud's overall ability with other clouds in order to identify the strengthes and weaknesses. The data would be normalized to the range from 0 to 5 in order to compare with the AWS cluster's performance with 2 m1.medium instances.

The inputs are 2 1024 $\times$ 1024 matrices and the output should be the matrix consisting of the multiplication of the corresponding elements of the original 2 matrices. The matrix multiplication job is submitted as a mapreduce streaming task where the mapper and reducer are specified by the python files (*MatMulMapper.py* and *MatMulReducer.py*)

a. Experimental process:

I fix the number of mapper to 1 and change the number of reducers from 1 to 3 in order to measure the speedup (of 3 reducers nodes compared to that of 1 node) and scalability of the AIRS cloud. The job is submitted in the following command:

```
  mapred streaming -D mapred.map.tasks=$MAP_NUM$ -D mapred.reduce.tasks=$RED_NUM$
-input /dataspace/teamY/accountX/input   -output /dataspace/teamY/accountX/output
-mapper "python MatMulMapper.py"   -reducer "python MatMulReducer.py"   -file
/home/accountX/matrix/MatMulMapper.py   -file
/home/accountX/matrix/MatMulReducer.py
```

After the job had completed, I collected the results from *output* folder in HDFS and analyze them based on the terminal output results.

b. Results and discussions:

Denote P_{S_u}, P_S, P_B, P_{QoS} to be the scores that AIRS cloud earned with respect to speedup, scalability, bandwidth and QoS.

Figure 11 shows how the speedup of the matrix multiplication change with respect to the number of reducers used.

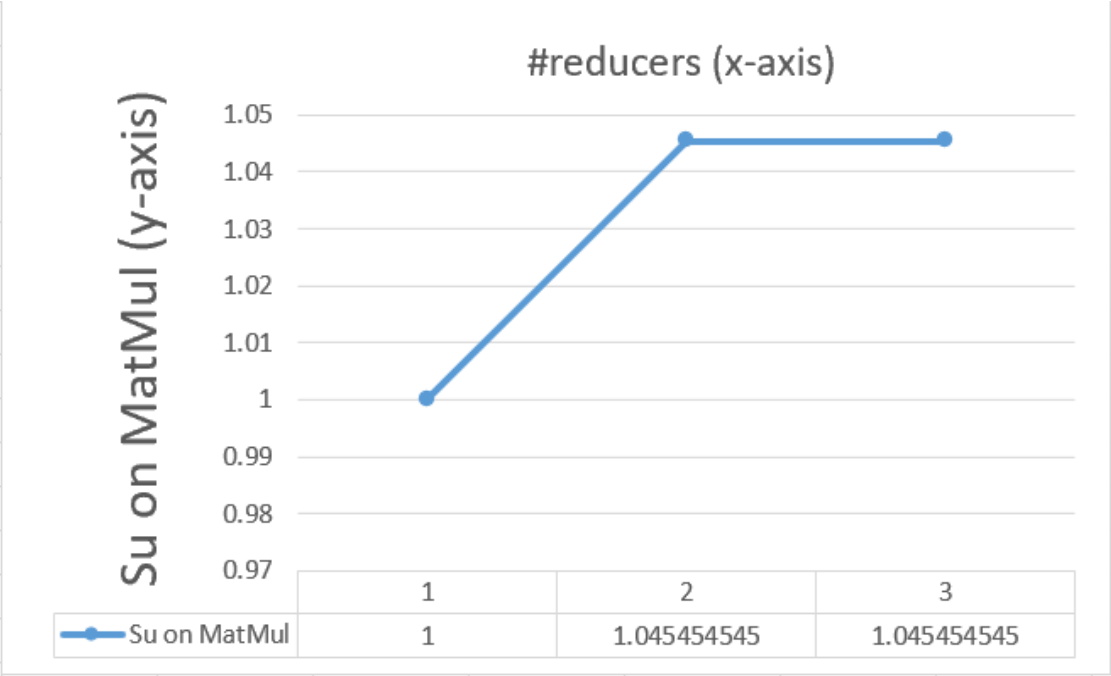


Figure 11: The change of S_u with respect to # reducers

Since the maximum possible speedup for this experiment set is 3 (corresponding to the maximum number of reducers), I use the portion the average speedup on different experiments can reach compared to the maximum to characterize the score:

$$P_{S_u} = \frac{E(S_u)}{3} \times 5$$

Figure 12 shows how the scalability of the AIRS cloud cluster on the matrix multiplication task changes with respect to the number of reducers used.

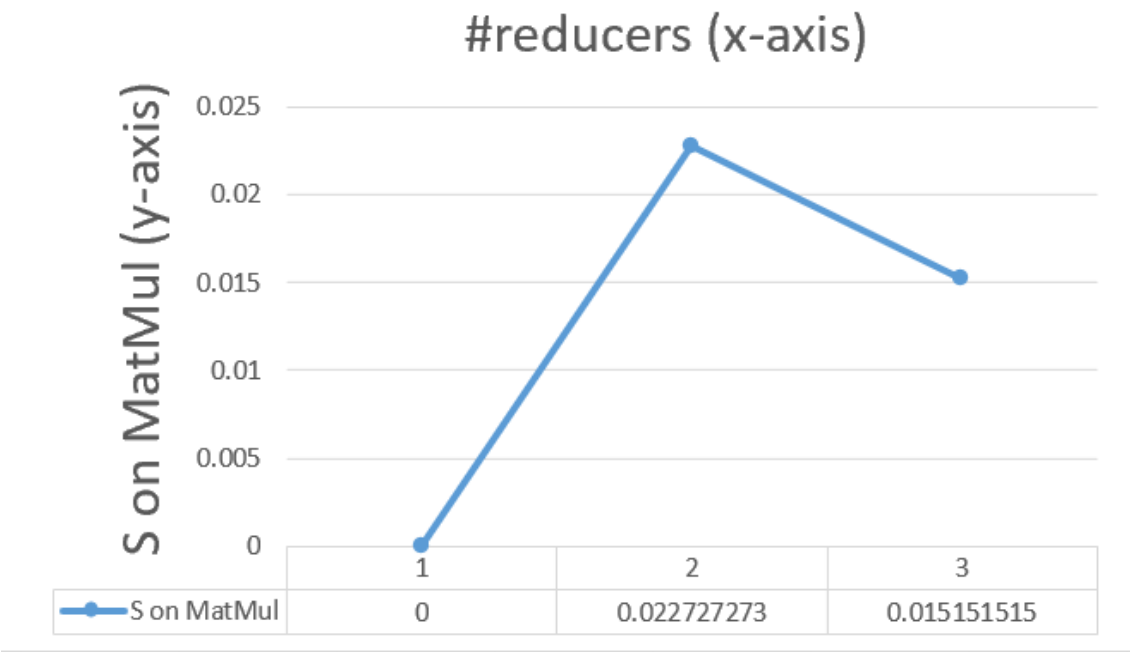


Figure 12: The scalability change with respect to # reducers

Since it is assumed that ideally the AIRS cloud can have a speedup which exactly coincides with the scale-outs, I set the maximum possible scalability to be 1.

$$P_S = \begin{cases} \frac{E(S)}{1} \times 5, & E(S) < 1 \\ 5, & otherwise \end{cases}$$

Figure 13 shows the measured bandwidth (bytes/s) of the AIRS cloud cluster on the matrix multiplication task under the circumstances of different number of reducers used.

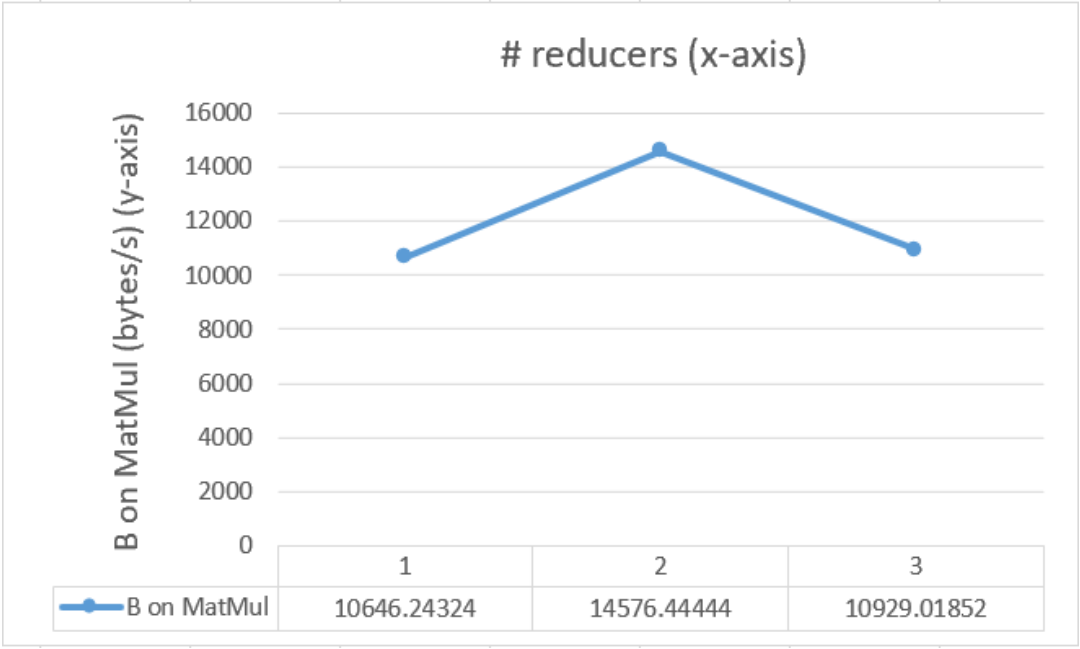


Figure 13: The bandwidth change with respect to # reducers

The score is directly propotional to the score earned by the benchmark cluster, which is to be compared by the AIRS cloud.

$$P_B = \frac{E(B)}{B_{BenchmarkCluster}} \times P_{B_{BenchmarkCluster}}$$

Figure 14 shows the total number of shuffle errors on the AIRS cloud cluster on the matrix multiplication task under the circumstances of different number of reducers used.

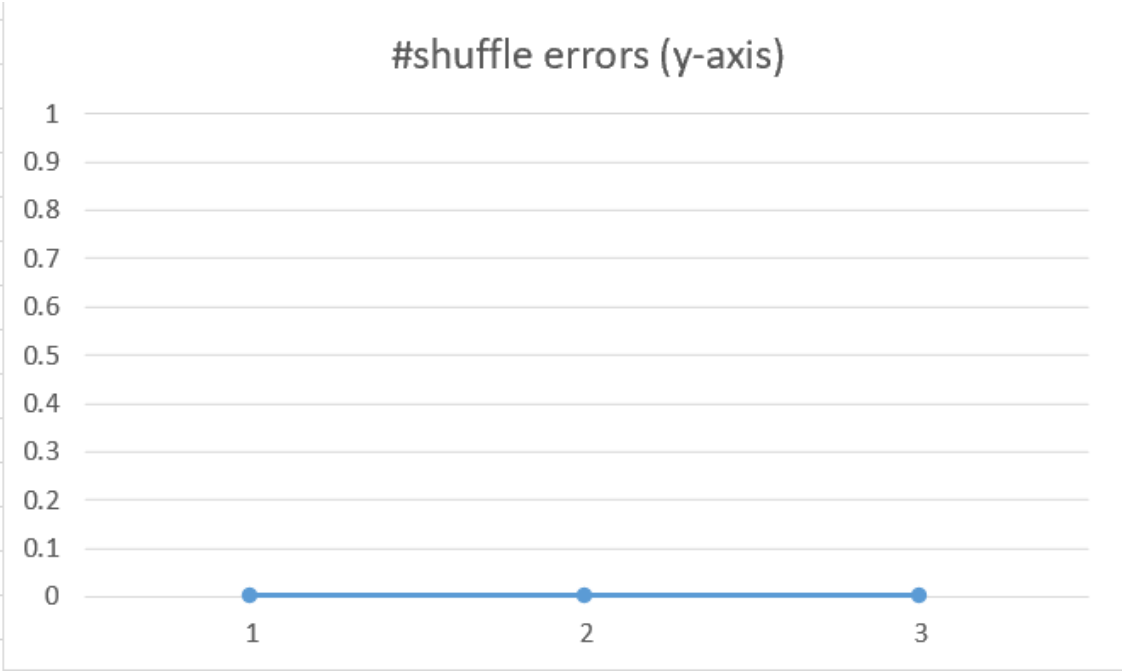


Figure 14: # shuffle errors change with respect to # reducers

The corresponding score would be characterized by the

$$P_{QoS} = 5 - E(QoS)$$

After that, I drew a radar chart in figure 15 comparing the results of the AIRS cloud and a cluster consisting of 2 m1.median instances under QoS, Bandwidth, Scalability and Speedup, as shown in figure 15. I find that the AIRS cloud is strong at providing reliable QoS, which is characterized essentially by the zero occurance of wrong maps, reduces and I/O errors when running matrix multiplication. It indicates the AIRS cloud's high availability on applications. However, it can be seen that the AIRS cloud has an insatisfactory scalability in that it's speedup doesn't match the number of nodes scaled. The bandwidth for AIRS cloud is also relatively insatisfactory compared to the little AWS cluster since it has a lower network transfer rate. This might be further due to the network traffic limitations imposed by AIRS. In addition, the area captured by the AWS cluster in the radar chart is larger than that of AIRS Cloud, indicating that the AIRS Cloud has a worse overall performance.

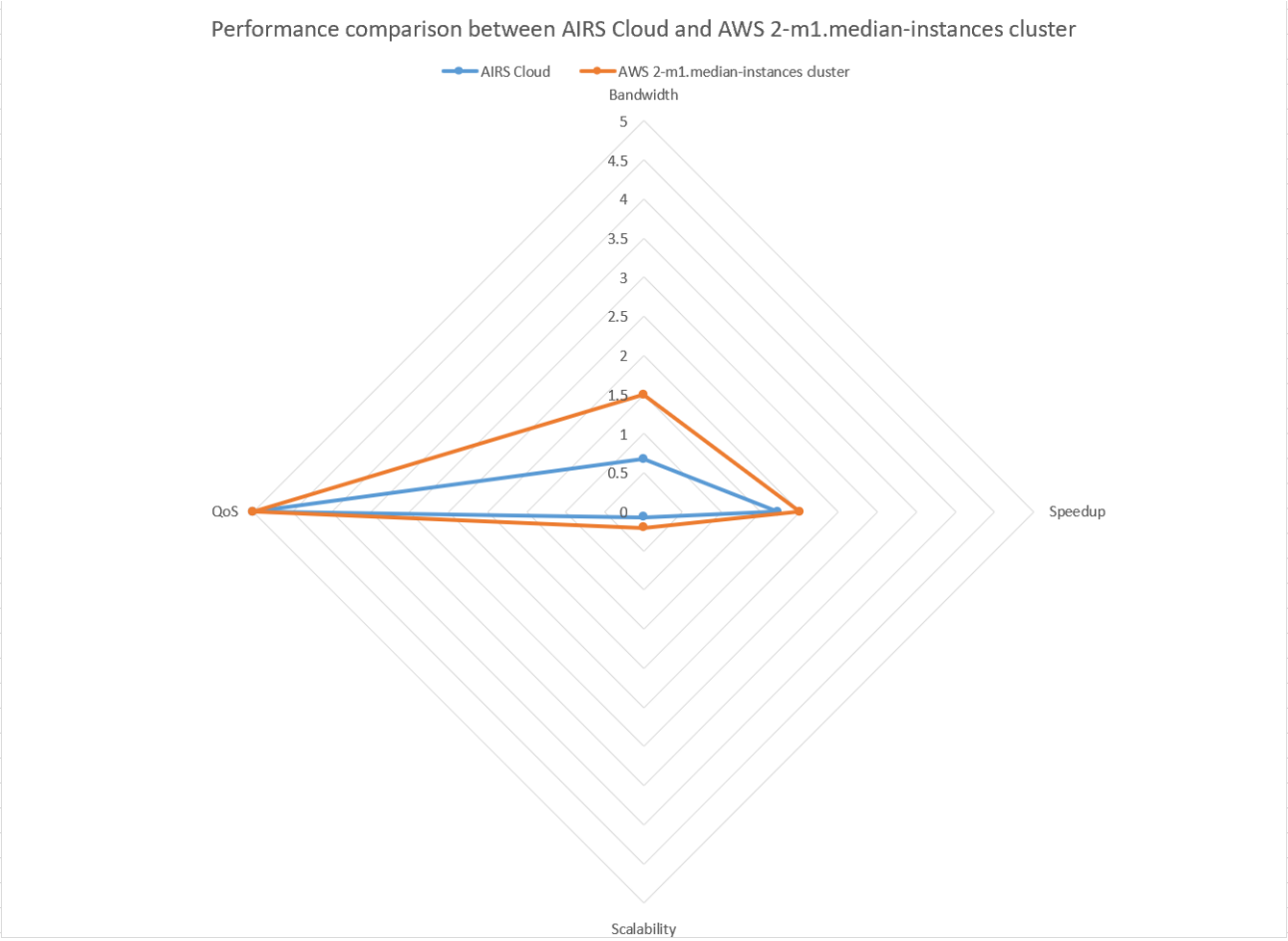


Figure 16: Radar chart for performance comparison