Benchmarking Joyent SmartDataCenter for Hadoop MapReduce and MPI Operations

Weiliang Luo(valiantluo@gmail.com), Nima Golpavar(nima.golpavar@gmail.com), Carlos Cardenas[+] (carlos.cardenas@joyent.com), Anthony T. Chronopoulos(atc@cs.utsa.edu)
Department of Computer Science
University of Texas at San Antonio
1 UTSA Circle, San Antonio, Texas 78249, U.S.A.
and
[+]Joyent Inc.
1 Embarcadero Center, San Francisco, CA 94111, U.S.A.

***Abstract*—**Cloud Computing is an ever-growing paradigm shift in computing allowing users commodity access to compute and storage services. As such cloud computing is an emerging promising approach for High Performance Computing (HPC) application development. Automation of resource provision offered by Cloud computing facilitates the eScience programmer usage of computing and storage resources. Currently, there are many commercial services for compute, storage, network and many others from big name companies. However, these services typically do not have performance guarantees associated with them. This results in unexpected performance degradation of user's applications that can be somewhat random to the user. In order to overcome this, a user must be well versed in the tools and technologies that drive Cloud Computing. One of the state of the art cloud systems, Joyent SmartDataCenter, is a cloud system that provides virtual machines (and their processes) the ability to burst CPU capacity automatically and thus is suitable for HPC applications. To help HPC developers, we present a set of Hadoop MapReduce and MPI benchmarks for FlexCloud (a SmartDataCenter installation). Our benchmarks show that this cloud system offers scalable performance for HPC environments.

Keywords- Cloud computing, Hadoop MapReduce, MPI, benchmarks

Introduction

Over the last several years, Cloud Computing has taking off in terms of usage. These systems provide compute and storage services that offer: scalability, flexibility, reliability, and on-demand computing.

There are several commercial cloud providers such as: AmazonEC2, Microsoft Azure, Salesforce Service Cloud and Google Cloud. Also, there are some open source cloud projects for research and development: OpenStack, Eucalyptus, CloudStack and Ganeti (see [1] and references there in).

We next reviewed some examples of recent research results for cloud systems. In [2], a provisioning technique that automatically adapts to workload changes related to applications with Quality of Services (QoS) in large, autonomous, and highly dynamic environments is proposed. [3] extends Grid workflow middleware to compute clouds in order to speed up executions of scientific applications. We are observing a trend of Cloud Computing being used to solve computationally intensive jobs in the HPC domain. They are becoming an alternative to private clusters and grids. Parallel production environments on cloud environments may introduce performance overhead for the demanding scientific computing workloads due in part to the resource sharing of the several independent virtual machines. The performance of Cloud Computing services for scientific computing workloads is studied in [4]. Cloud systems offer a utilities based model that facilitates working with large amounts of compute power without the need to own a parallel distributed system [5].

Scientific computing applications with varying compute and storage requirements are suitable for the pay-as-you-go model that Cloud Computing provides since resources can be provisioned when needed. Some of these advantages have been published in [5],[6],[7], and [8]. Cloud computing platforms provide a near infinite resource pool of resources by means of virtualization technology [9].

For HPC applications, this means using clusters of virtual machines. These virtual machines can share the same physical hardware with varying compute loads. Each cloud system uses their own resource contention algorithm for providing fairness to the shared resources: CPU, memory, disk, and NIC. In SmartDataCenter, Joyent uses what is called a fair-share scheduler which balances compute loads on a system based on contention and priority [13].

Due to limited resources and to get the full life of a machine, some Cloud system platforms have heterogeneous compute environments like AWS EC2 that provides both Intel Xeon and AMD Opteron while others have varying generations of CPUs. Even in the case of homogeneous hardware, the virtualized cluster provisioned to an HPC user shares resources with other users. This means that the virtualized cluster may act as a heterogeneous computing environment at running time.Thus, the heterogeneity should be taken into

account to improve resource utilization and reduce load imbalance in order to achieve efficient performance.

In order to overcome this, MapReduce, a general concurrent programming framework for scheduling jobs [10], is a sight for sore eyes. However, the MPI concurrent programming model may yield in general higher performance than MapReduce. Previous research reported that the performance on virtual machines is lower than physical systems [11] and [12]. These papers analyzed message passing (MPI) parallel applications on different cloud systems and reported that communication overhead is a substantial slowdown factor. [15] presents a set of benchmarks for storage operations with the Azure cloud system. We are motivated by [15] in our work to benchmark Joyent SmartDataCenter for scientific computations.

We present computational and communication experiments on Joyent SmartDataCenter (FlexCloud). We use the Apache Hadoop for MapReduce and OpenMPI's MPI implementation. Our experiments demonstrate the scalability of this cloud system.

The rest of the paper is organized as follows. In Section II, we present Joyent SmartDataCenter cloud. In Section III, we present our Hadoop MapReduce experiments. In Section IV, we present our MPI experiments. Section V contains conclusions and future work.

JOYENT SMARTDATACENTER

We use Joyent SmartDataCenter operated by the of Institute for Cyber Security (ICS) at the University of Texas at San Antonio. The ICS FlexCloud is one of the first dedicated Cloud Computing academicresearch environments. It offers significant capacity andsimilar design features found in Cloud Computing providers, including robust compute capability and elastic infrastructuredesign. FlexCloud highlights currently include:

_ Racks of Dell R410, R610, R710, and R910s consisting of 748 processing cores, 3.44TB of RAM, and 144TB of total storage.

_Redundant 10GB network connectivity provides high performance access between all nodes.

_Powered by Joyent SmartDataCenter [13] providing the highest performance virtualization and analytics. And Joyent SmartOS provides a combination of hardware and operating system virtualization to support efficient, reliable and high performing cloud computing.

– Joyent uses the HPC model of management: one headnode PXE boots compute nodes.

– SmartOS is a RAM disk based image (nothing stored on disk) which makes it very easy to upgrade headnodes/computenodes.

– SmartOS uses the disks on the local nodes to back TheSmartMachines and Virtual Machines usingZFS.

– SmartOS has DTrace which allows for monitoringall VMs with little overhead for maximum observability. SmartOS has the best multitenant defenses to preventone tenant from affecting others on the box.

For all our experiments, we initialize 32 virtual machine instances,each with one virtual core instance, 1GB memory and10GB storage. Each instance is loaded with an Ubuntu Linux 10.04 image.

HADOOP MAPREDUCE EXPERIMENTS

Hadoop is an open source software platform from Apache [16].It consists of the following components: Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data. Hadoop MapReduce is a YARN-based system for parallel processing of large data sets [17].

We installed the Apache Hadoop on the Flexcloud and implemented two benchmarks:

a. A matrix times matrix multiplication benchmark (a block matrix algorithm proposed in [18]).

b. A Data mining benchmark using files of crimes data in Austin, Texas.

a. The Matrix times matrix multiplication benchmark

i. Matrix A is divided by row blocks and matrix B is divided by column blocks (as shown in Fig. 1).

Fig.1:Matrix multiplication

ii. Design in MapReduce: (as shown in Fig. 2), splits multiplications (keys) amongst mappers (tasks) which are subsequently summed by the reducers (tasks).

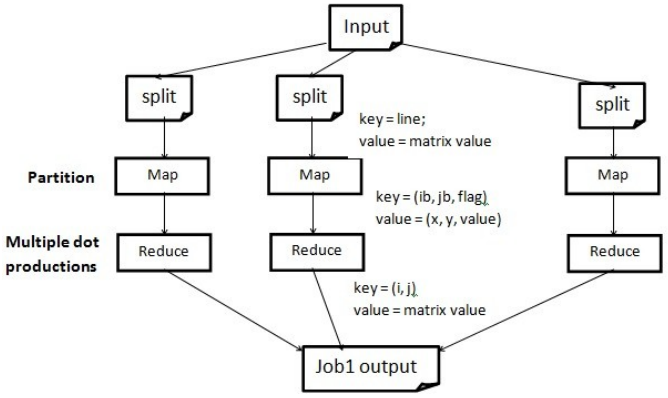

Fig.2: MapReduce process

iii. Performance Evaluation

We ran this experiment with square matrices of size 4096*4096. We used 32, 16, 8, 4, 2 and 1 VMs, with one mapper/reducer running inside each VM. Our performance results and speedups are plotted in the following graphs (Fig. 3 and Fig. 4).

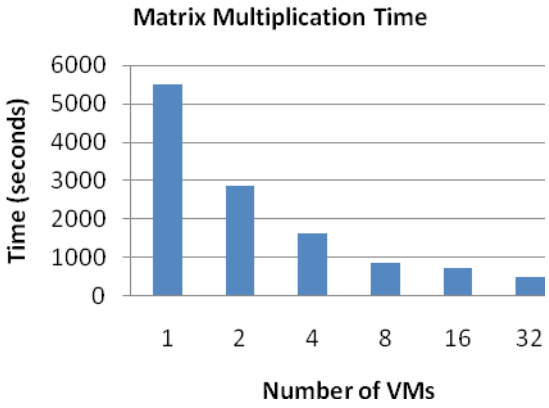

Fig.3: Matrix multiplication execution times

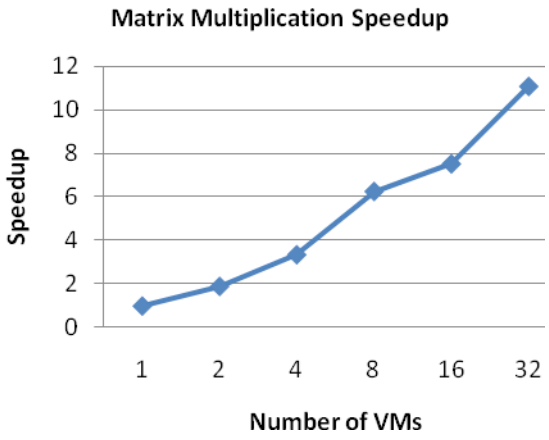

Fig.4: Matrix multiplication speedups

b. Data mining benchmark

We use the data base of the city of Austin, Texas, U.S.A. Police reports. The problem is to answer the question: (i) *Where is most of the crime happening in Austin?*To answer this question, we use 1 MapReduce job to process the data. In the mapper, we partitioned the crime records based on the longitude and latitude into 36 partitions (Fig. 5). For each occurrence of crime, we emit a key and value pair with key equals 1 to 36 (representing 36 different areas) and value equals 1. In the reducer, we sum up the values with the same key. This way, we get the number of crimes happened in each area. Our approach allows for 32 mappers and 32 reducers.

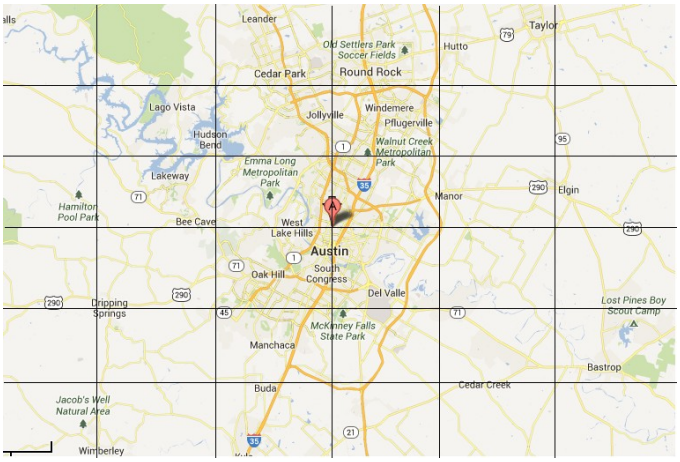

Fig.5: Data mining partition

i. **Performance with different node numbers and input sizes**

We measured the performance on question (i), with different data sizes. Sincethe size of the actual file is fixed, we changed the input size by duplicating the input data many times. In our experiment the data wasduplicated 512 times ofthe original data. These large files were used to run on 1, 2, 4, 8, 16 and 32

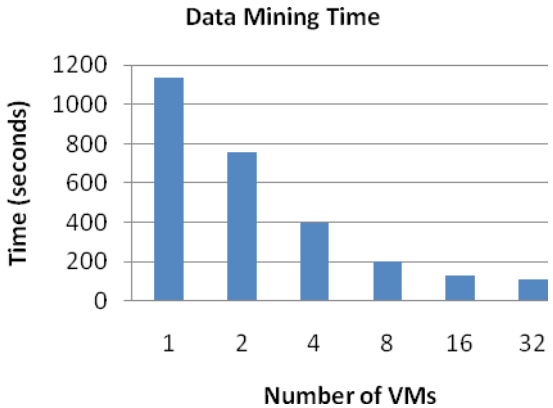

Fig.6: Data mining execution times

nodes. Performance results are shown in Figures 6 and 7.

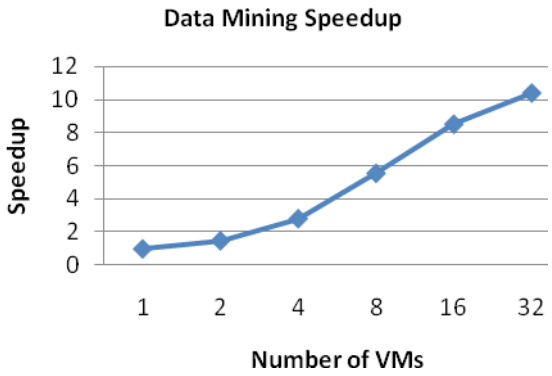

Fig.7: Data mining speedups

We can observe that the best speed up is for 32VMs which is 10.39.

MPI EXPERIMENTS

In these experiments we ran the basic MPI communication operation and also matrix timesmatrix,matrix times vector multiplication algorithms.

MPI_Bcast

Broadcasts (sends) a message from the process with rank "root" to all other processes in the group.

MPI_Bcast(&buffer,count,datatype,root,comm)

MPI_Reduce
Applies a reduction operation on all tasks in the group and places the result in one task.

MPI_Reduce(&sendbuf,&recvbuf,count,datatype,op,root,comm)

MPI_Allreduce
Applies a reduction operation and places the result in all tasks in the group. This is equivalent to an MPI_Reduce followed byanMPI_Bcast.

MPI_Allreduce(&sendbuf,&recvbuf,count,datatype,op,comm)

MPI_Scatter
Distributes distinct messages from a single source task to each task in the group.

MPI_Scatter(&sendbuf,sendcnt,sendtype,&recvbuf,recvcnt,recvtype,root,comm)

MPI_Gather
Gathers distinct messages from each task in the group to a single destination task.This routine is the reverse operation of MPI_Scatter.
MPI_Gather(&sendbuf,sendcnt,sendtype,&recvbuf, recvcount,recvtype,root,comm)

MPI_Allgather
Concatenation of data to all tasks in a group.
Each task in the group, in effect, performs a one-to-all broadcasting operation within the group.

MPI_Allgather(&sendbuf,sendcount,sendtype,&recvbuf,recvcount,recvtype,comm)

MPI_Alltoall
Each task in a group performs a scatter operation, sending a distinct message to all the tasks in the group in ordered by the index. [14]

MPI_Alltoall(&sendbuf,sendcount,sendtype,&recvbuf,recvcnt,recvtype,comm)

Timing results for all of the functions (program size is 128*1024) (shown in Table 1).

Program	Time(secs),2vms	Time(secs),32vm	Speedup
Bcast	0.023991703987	0.038851094246	1.65
Allgather	0.01853199005	0.184243488312	9.945
Alltoall	0.018950009346	0.121125411987	6.4
Gather	0.010587978363	0.038110089302	3.8
Reduce	0.030606102943	0.083671116829	2.73
Scatter	0.009745192528	0.026290607452	2.6
Allreduce	0.038705587387	0.124320793152	3.2
Matrix*vector	0.283890	0.037872	7.64
Matrix*Matrix	191.404235	11.952213	16.01

Table 1: times for MPI collective communications function functions

We have also implemented the following master-worker matrix times matrix product algorithm. The master creates matrices A and B and sends rows of A and columns of B to workers who multiply them and return the sub matrices of matrix C.

Matrix-Matrix Multiplication (Master)

```
NRA 1024       /* number of rows in matrix A */
NCA 1024       /* number of cols in matrix A */
NCB 1024       /* number of cols in matrix B */
/******* Master process **********/
    a[NRA][NCA],   /* matrix A to be multiplied */
    c[NRA][NCB];   /* result matrix C */
    for i:=0 → NRA do
      for j:=0 → NCA do
        a[i][j]:= double(i+j)
    end for
    for i:=0 → NCA do
      for j:=0 → NCB
        b[i][j]:= double(i*j)
     end for
/* Send matrix data to the worker processes; only once */
    for dest:=1 → numworkers do
      MPI_Send(&offset,1,MPI_INT,dest,mtype, MPI_COMM_WORLD)
      MPI_Send(&rows,1,MPI_INT,dest,mtype, MPI_COMM_WORLD)
      MPI_Send(&a[offset][0],count,MPI_DOUBLE,dest,mtype,MPI_COMM_WORLD)
      MPI_Send(&b[0][0],count,MPI_DOUBLE,dest,mtype,MPI_COMM_WORLD)
    end for
   /* Start MPI timer here*/
   /* Wait for results from all worker processes */
     for i:=1 → numworkers do
        source := i
        MPI_Recv(&offset,1,MPI_INT,source,mtype,MPI_COMM_WORLD,&status)
        MPI_Recv(&rows,1,MPI_INT,source,mtype,MPI_COMM_WORLD,&status)
        count := rows*NCB
       MPI_Recv(&c[offset][0],count,MPI_DOUBLE,source,mtype,MPI_COMM_WORLD,&status);
      end for
   /*Calculating the total time here*/
  /* End of master */
```

Matrix-Matrix Multiplication (Worker)

```
/************ Worker process *************/
  MPI_Recv(&offset,1,MPI_INT,source,mtype,MPI_COMM_WORLD,&status)
  MPI_Recv(&rows,1,MPI_INT,source,mtype,MPI_COMM_WORLD,&status)
  count := rows*NCA
  a_w[row_max][NCA]   /* matrix A to be multiplied */
  c_w[row_max][NCB]   /* result matrix C */
  MPI_Recv(&a_w,count,MPI_DOUBLE,source,mtype,MPI_COMM_WORLD,&status)
  count := NCA*NCB
  MPI_Recv(&b,count,MPI_DOUBLE,source,mtype,MPI_COMM_WORLD,&status)
     for k:=0 → NCB       /* multiply worker part */
       for i :=0 → rows
         c_w[i][k] := 0.0
         for j:=0 → NCA
           c_w[i][k] := c_w[i][k] + a_w[i][j] * b[j][k]
     end for
    MPI_Send(&offset,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD)
    MPI_Send(&rows,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD)
    count := rows*NCB
    MPI_Send(&c_w,count,MPI_DOUBLE,MASTER,mtype,MPI_COMM_WORLD)
 /* End of worker */
```

We also implemented and ran a similar (master-worker) algorithm for matrix*vector product computation. In our experimentwe choose the configuration of VMs with the total number of 32, 16, 8, 4 and 2, sothat half of the VMs for both programs are from each of the twoseparate physical machines.

We next present performance results with varying number of VMs, shown in Figures 8,9,10,11.

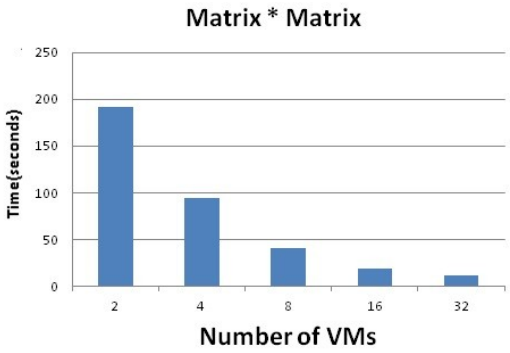

Fig.8: Execution Times

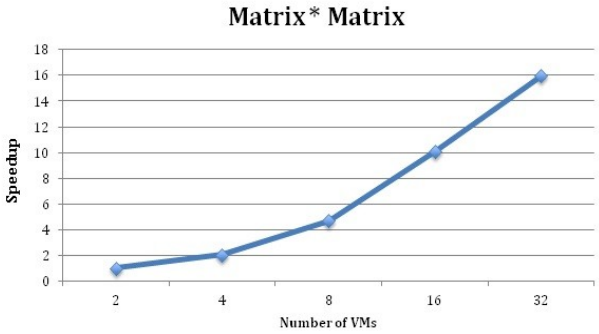

Fig.9: Speedups

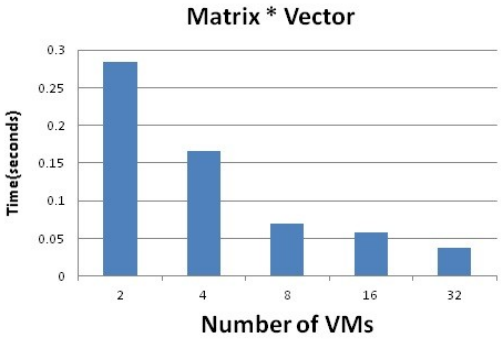

Fig.10: Execution Times

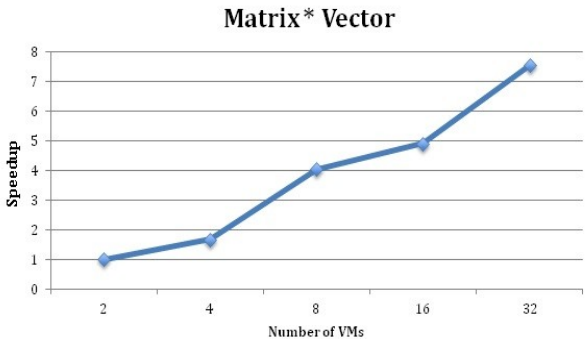

Fig.11: Speedups

We observe here good scalability with speed up for 32 VMs over 2 VMs. For Matrix* Matrix the speed up is 16.01 and for Matrix*Vector is 7.64.

CONCLUSIONS AND FUTURE WORK

In this paper we presented a set of computation and communication benchmarks based on Hadoop MapReduce and MPI for Joyent FlexCloud system. Our results show the scalability of the cloud system and are useful to scientific computing applications. In future, we plan to investigate the performance of storage services combined with computations for the FlexCloud system.

Acknowledgements

We gratefully acknowledge the following:
(i) support by NSF grant (HRD-0932339) to the University of Texas at San Antonio; and (ii) time grants to access the Facilities of the Institute for Cyber Security(ICS) of University of Texas at San Antonio.

REFERENCES

[1] K. Hwang, J. Dongarra, and G. C. Fox, Distributed and Cloud Computing: From Parallel Processing to the Internet of Things. Morgan Kaufmann, 2011.

[2] R. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and Qos in cloud computing environments," 2011 International Conference on Parallel Processing(ICPP), pp. 295–304, 2011.

[3] S. Ostermann, R. Prodan, and T. Fahringer, "Extending grids with cloud resource management for scientific computing," 2009 10th IEEE/ACM International Conference on Grid Computing, pp. 42–49, 2009.

[4] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," IEEE Transactions on Parallel and Distributed Systems, pp. 931–945, 2011.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available:http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[6] C. A. Lee, "A perspective on scientific cloud computing," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM,2010, pp. 451–459.

[7] G. Turcu, I. Foster, and S. Nestorov, "Reshaping text data for efficient processing on amazon ec2," in *Proceedings of the 19th ACMInternational Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 435–444. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851540

[8] A. Thakar and A. Szalay, "Migrating a (large) science database to the cloud," in *Proceedings of the 19th ACM International Symposiumon High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 430–434. [Online]. Available:http://doi.acm.org/10.1145/1851476.1851539

[9] J. E. Simons and J. Buell, "Virtualizing high performance computing," ACM SIGOPS Operating Systems Review, pp. 136–145, 2010.

[10] W.-C. Shih, S.-S. Tseng, and C.-T. Yang, "Performance study of parallel programming on cloud computing environments using MapReduce," 2010 International Conference on Information Science and Applications(ICISA), pp. 1–8, 2010.

[11] J. Ekanayake and G. Fox, "High performance parallel computing with clouds and cloud technologies," Proceedings of the first International Conference on Cloud Computing, pp. 20–38, 2010.

[12] C. Evangelinos and C. N. Hill, "Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2," Computability and Complexity in Analysis, pp. 159– 168, 2008.

[13] Joyent, http://joyent.com/.

[14] https://computing.llnl.gov/tutorials/mpi/

[15] Dinesh Agarwal and Sushil K. Prasad, "AzureBench: Benchmarking the Storage Services of the Azure Cloud Platform"2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

[16]

[17] Apache Hadoop, http://hadoop.apache.org/

[18] MapReduce: Simplified Data Processing on Large Clusters', Jeffrey Dean and Sanjay Ghemawat, OSDI '04: 6th Symposium on Operating Systems Design and Implementation, p. 137-149, 2004.

[19] 'A MapReduce Algorithm for Matrix Multiplication', John Norstad, Northwestern University, Academic and Research Technologies, http://www.norstad.org/matrix-multiply/