Parallel computing on cloud for massively intensive applications using mapreduce

A.Sree Lakshmi

Abstract— Recent advancements in data intensive computing for science discovery are fueling a dramatic growth in use of data-intensive iterative computations. Performance is an open issue in data intensive applications. To process large-scale datasets with high performance more resources and reliable infrastructures are required for spreading the data and running the applications across multiple machines in parallel Cloud computing basic platform can propose different requirements for storage and Computation in face of different users during its operation. Cloud computing has become the cost-effective solution for the increased demand for computing resources and services ,distributed data, by removing the the cost of building , operating and maintaining expensive physical resources and infrastructures . MapReduce is a programming model for parallel data processing widely used in Cloud computing environment. This paper discusses about MapReduce which is a cloud computing support to execute massively intensive applications parallel.

Index Terms— Parallel Comuting, cloud computing, Map reduce, Hadoop, data intensive computing,

I. INTRODUCTION

Cloud computing has emerged as a very popular computing paradigm offering on demand resource allocation and usage, lower costs, distributed storage and backup, and a usage based pricing model [1], [2]. Cloud computing is an evolution of various forms of distributed computing systems: from original distributed computing, parallel computing (cluster, to service-oriented computing (e.g., grid). Cloud computing extends these various distributed computing forms by introducing a business model, in which the nature of on-demand, self service and pay-by-use of resources is used. Cloud computing sells itself as a cost-effective solution and reliable platform. It focuses on delivery of services guaranteed through Service Level Agreements. The services can be application software—SaaS, development environments for developing applications—PaaS, and raw infrastructures and associated middleware —IaaS.

MapReduce is an attractive model for opportunistic computer resources and offers an ease-of-use programming paradigm for processing large data sets. MapReduce is an emerging programming model that simplifies parallel data processing [15]. Due to the feature of easy programming, the fault tolerance support, and the assumption of commodity hardware, MapReduce has been recognized as a compelling programming paradigm for non dedicated distributed computing environment to expand the scope of supported applications.

A.Sree Lakshmi, Associate Professor , Geethanjali College of Engineering and Technology, Hyderabad.

Cloud technologies such as Google MapReduce, Hadoop and Hadoop Distributed File System (HDFS), CGL-MapReduce nodes of clusters or large-scale data centers , as done in Google. To perform data processing the computations move to the data. GFS and HDFS which are distributed file systems allow Google MapReduce and Hadoop to access data via distributed storage systems built on heterogeneous compute nodes, while CGL-MapReduce and Dryad support reading data from local disks. The programming model is simple which enables better support for quality of services such as monitoring and fault tolerance.

Cloud computing is at the peak of the Gartner technology hype .There are several reasons why clouds should be important for large scale scientific computing

1) Clouds provide the large scale computer centers which is important to large scale science problems as well as those at small scale.
2) Clouds are cost effective approach to computing. Their architecture addresses the important fault tolerance issue explicitly.
3) Clouds are commercially supported and provide reasonably robust software without the sustainability difficulties seen from the academic software systems critical to much current Cyber infrastructure.
4) There are 3 major vendors of clouds (Google, Amazon, and Microsoft) and many other infrastructure and software cloud technology vendors including Eucalyptus Systems. This competition should ensure that clouds should develop in a healthy innovative fashion and following cloud standards [3]
5) There are many Cloud research, conferences and other activities with research cloud infrastructure efforts including OpenNebula[5], Eucalyptus[7],Nimbus[4], Sector/Sphere[6] and.

IJSER

6) Clouds offer "on-demand" and interactive computing that is more attractive than batch systems to many users.

II. OVERVIEW OF PARALLEL APPROACH

Parallel computing is a type of computational in which many programs are transferred instantly following the procedure that the huge problems are further divided in to smaller one and being solved concurrently. It is well known that the speedup of an application to solve large computational problems is mainly gained by the
Parallelization at either hardware or software levels or both. Software parallelism at component and system levels can be classified into two types: automatic parallelization of applications without modifying existing sequential applications and construction of parallel programming models using various software technologies to describe parallel algorithms and then match applications with the underlying hardware platforms. Since the nature of auto-parallelization is to recompile a sequential program without the need for modification, it has a limited capability of parallelization on the sequential algorithm itself. Mostly, it is hard to directly transform a sequential algorithm into parallel ones. While parallel programming models try to address how to develop parallel applications and therefore can maximally utilize the parallelization to obtain high performance, it does need more development effort on parallelization of specific applications.

In general, three considerations when parallelizing an application include:

- How to distribute workloads or decompose an algorithm into parts as tasks?
- How to map the tasks onto various computing nodes and execute the subtasks in parallel?
- How to communicate and coordinate subtasks on those computing nodes.

There are mainly two common methods for dealing with the first two questions: data parallelism and task parallelism.
Data parallelism represents workloads are distributed into different computing nodes and the same task can be executed on different subsets of the data simultaneously.
Task parallelism means the tasks are independent and can be executed purely in parallel. There is another special kind of the task parallelism is called 'pipelining'. A task is processed at different stages of a pipeline, which is especially suitable for the case when the same task is used repeatedly. The extent of parallelization is determined by dependencies of each individual part of the algorithms and tasks.

For one computational process, loop structure is the one that appears most frequent in program design and also occupies system computation resource mainly. Frequently, during the running of a program, most of the time is spent on executing loop program. The increase of the layer number of loop and nest makes the time for computing in exponential rise, which becomes the main computing bottleneck during the running of a program. Therefore, doing research on the problem of parallelism of loop program is one most import aspect for us to do the job about the parallelism of program.

Loop Structure 1:

```
for (i=0;i<1000;i++)
{
…;
}
```

Loop Structure 2:

```
for (i=0; i<1000;i++)
for (j=0; j<1000; j++)
for (k=0;k<1000; k++)
{
…;
}
```

Because the two loops above differ by two layers of nests their execution time differ from each other about one million times. Hence, when doing tasks partition, to process loop structure is important in the process of parallelism. The correlation between data has very important impact on parallelism.

When we do research on the parallelism of loop program we must analyze the dependency between computational data[17] [18] and thus discuss the method of data partition when executing computation. As long as there is dependency between data, we have to pay more attention in the process of the parallelism of program. Analyzing the dependency between statements in a computing program is called dependency analysis.

III. MAP REDUCE MODEL

MapReduce[21] is a system and method for efficient large-scale data processing presented by Google in 2004 to cope with the challenge of processing very large input data generated by Internet-based applications. Since its introduction, MapReduce has is applicable to a wide range of domains, including scientific simulation, image retrieval and processing, machine learning and data mining, financial analysis, log file analysis, blog crawling, language modelling, machine translation and bioinformatics.

One can describe the difference between MPI and MapReduce as follows. In MapReduce multiple map processes are formed -- typically by a domain(data) decomposition familiar from MPI -- these run asynchronously typically writing results to a file system that is consumed by a set of reduce tasks that merge parallel results in some fashion. This programming model implies straightforward and efficient fault tolerance by re-running failed map or reduce tasks. MPI addresses more complicated problem architecture with iterative compute--communicate stages with synchronization at communication phase. This synchronization means that all the processes wait if one is failed or delayed. This inefficiency is not present in MapReduce where resources are released when individual map or reduce tasks complete. MPI supports general (built in and user defined) reductions therefore MPI can be used for applications of the MapReduce style. However MapReduce offers greater fault tolerance and user friendly higher level environment largely stemming from the coarse grain functional programming model implemented as side-effect free tasks. MPI supports multiple Map-Reduce stages but MapReduce just one. Correspondingly MapReduce supports applications that have the loose coupling while classic HPC [20] supports more tightly coupled applications

MapReduce is one of the most important programming models for Cloud computing environments, supported by leading Cloud providers such as Amazon, with its Elastic MapReduce service, and Google itself, which recently released a Mapper API for its App Engine.

The MapReduce abstraction is inspired by the *map* and *reduce* primitives present in Lisp and other functional languages. A user defines a MapReduce application in terms of a map function that processes a (key, value) pair to generate a list of intermediate (key, value) pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

Map: : (key1; value1) ! arraylist (key2; value2)

reduce: : (key2; list(value2)) ! list (value3)

Current MapReduce implementations,

like Hadoop and Google's MapReduce , are based on a master-slave architecture. User node submits a job to a master node that selects idle workers and assigns a map or reduce task to each one.

The master node returns the result to the user node when all the tasks have been completed. The failure of a worker is substituted by another worker and re-executing its task, while master failures are not explicitly managed as they are considered that failures are unlikely in reliable computing environments.

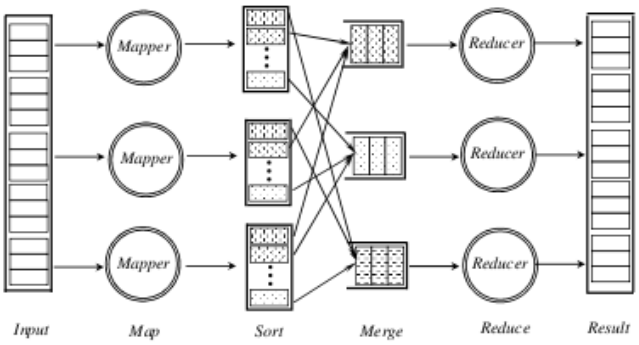

Fig. 1. Computation of MapReduce

The entire architecture (Figure 1) is grouped to form 2 independent modules.

Phase 1 (Map and Sort): Each key-value pair is fed to the map function and results are sent to a global buffer. The buffer comprises of a definite number of buckets, each one for a different position. A threshold is chosen and , the buffer data is flushed on to the secondary storage once the output exceeds the threshold. Each bucket is sorted in the memory before the buffered results are written into the disk,. Quick Sort is generally used. The data written on to the disk is sorted by key.

Phase 2 (Merge and Reduce) : Merge operation starts as and when it gets inputs. The results are grouped on the basis of key and the values are clubbed together on a list. In case of collision in buckets, the new value is append towards the end of the value list for that key. Heap Sort is commonly used. The Reduce stage iterates over all keys and applies the user defined reduce function on them. The results obtained are written back to the disk.

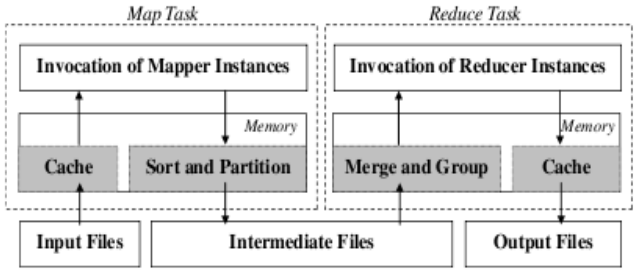

Fig. 2. DataFlow in MapReduce

MapReduce uses a Scheduling mechanism to optimize node utilization. Nodes are constantly monitored by the Scheduler to make sure that no tasks are stalled or show erroneous behavior. Nodes tend to be slow when jobs are heterogeneous or dominating users interfere with each other. Free Resources in clouds are shared by the resource owner as well as by other users who have idle resources.

A static scheduling mechanism in this situation might lead to soft failure, where a MapReduce execution has to abort computation due to domination by its owner. Static Scheduling, hence, is not preferred in Enterprise Clouds.

Realtime scheduling is highly efficient as it constantly picks out nodes that are idle and assigns new tasks to them. The Scheduler thus starts off by dispatching the map tasks to all nodes simultaneously. Whenever a reduce task is ready, it will be scheduled on the basis of the current status of resources.

MapReduce, being a parallel computing model, is both time consuming and error prone. It is not possible to incorporate a parallel computation model for all applications.Communication, Coordination and synchronization between the nodes are prime requirements. Most of the parallel programs are asynchronous and it is pretty hard to analyze the interaction between the machines. We now look at two programming models which were actually proposed for Grid-based applications, but can adapt seamlessly to Cloud Infrastructures

The current scientific computing is vastly populated by the growing set of data-intensive computations that require enormous amounts of computational as well as storage resources and novel distributed computing frameworks. The pay-as-you-go Cloud computing model provides an option for the computational and storage needs of such computations.

The new generation of distributed computing frameworks such as MapReduce focuses on catering to the needs of such data-intensive computations.Iterative computations are at the core of the vast majority of scientific computations. Many important data intensive iterative scientific computations can be implemented as iterative computation and communication steps, in which computations inside an iteration are independent and are synchronized at the end of each iteration through reduce and communication steps, enabling it for individual iterations to be parallelized using technologies such as MapReduce. The growth in number of data intensive iterative computations as well as importance is driven partly by the need to process massive amounts of data and partly by the emergence of data intensive computational fields, such as scientific applications, web mining bioinformatics, chemical informatics.

A lot of progress has been made with the MapReduce framework originally developed for information retrieval -- a really enormous data intensive application. Initial research shows this is a really promising approach to much scientific data analysis.

MapReduce programming models offer better fault tolerance and dynamic flexibility than MPI and so should be used in loose coupling problems in preference to MPI. Parallel BLAST is a good example of a case where Generalizing from this, clouds are more important for data intensive applications than classic simulations as latter are very sensitive to synchronization costs which are higher in clouds than traditional clusters.

MapReduce and Clouds can be used for some of the applications that are most rapidly growing in importance. Their support seems essential if one is to support large scale data intensive applications. More generally a more careful analysis of clouds versus traditional environments is needed to quantify the simplistic analysis given above.

There is a clear algorithm challenge to design more loosely coupled algorithms that are compatible with the map followed by reduce model of MapReduce or more generally with the structure of clouds. This could lead to generalizations of MapReduce which are still compatible with the cloud virtualization and fault tolerance features.

IV. A CASE STUDY: THE GREPTHEWEB APPLICATION

An application called *GrepTheWeb* at Amazon illustrates the power and the appeal of cloud computing. The application allows a user to define a regular expression and search the web for records that match it.

GrepTheWeb is analogous to the *grep* Unix command used to search a file for a given regular expression. This application performs a search of a very large set of records attempting to identify records that satisfy a regular expression. The source of this search is a collection of document URLs produced by the *Alexa Web Search*, a software system that crawls the web every night. The inputs to the applications are a regular expression and the large data set produced by the web crawling software; the output is the set of records that satisfy the expression. The user is able to interact with the application and get the current status. The application uses message passing to trigger the activities of multiple controller threads which launch the application, initiate processing, shutdown the system, and create billing records.

Performing a regular expression against millions of documents is not trivial. Large time of processing could be because of different factors like

- complex Regular expressions
- Dataset could be large, even hundreds of terabytes
- Unknown request patterns

GrepTheWeb uses *Hadoop MapReduce*, an open source software package that splits a large data set into chunks, distributes them across multiple systems, launches the processing, and, when the processing is complete, aggregates the outputs from

different systems into a final result. *Apache Hadoop* is a software library for distributed processing of large data sets across clusters of computers using a simple programming model.

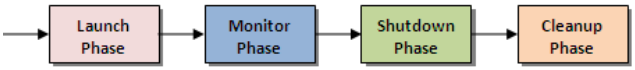

Fig. 3: workflow of GrepTheWeb

The details of the workflow of *GrepTheWeb* are captured in Figure 3 and consist of the following steps :

1. *The start-up phase:* create several queues - launch, monitor, billing, and shutdown queues; start the corresponding controller threads. Each thread polls periodically its input queue and when a message is available, retrieves the message, parses it, and takes the required actions.

2. *The processing phase:* it is triggered by a StartGrep user request; then a launch message is enqueued in the launch queue. The launch controller thread picks up the message and executes the launch task; then, it updates the status and time stamps in the Amazon *Simple DB* domain. Lastly, it enqueues a message in the monitor queue and deletes the message from the launch queue. The processing phase consists of the following steps:

(a) Firstly the launch task starts Amazon *EC2* instances: it uses a Java Runtime Environment pre-installed Amazon Machine Image (AMI), deploys required *Hadoop* libraries and starts a Hadoop Job (run Map/Reduce tasks).

(b) *Hadoop* runs map tasks on Amazon *EC2* slave nodes in parallel: a map task takes files from Amazon *S3*, runs a regular expression and writes locally the match results along with a description of up to five matches; then the combine/reduce task consolidates the output by combining and sorting the results

(c) Final results are stored on Amazon *S3* in the output bucket.

3. *The monitoring phase:* the monitor controller thread retrieves the message left at the beginning of the processing phase, validates the status/error in Amazon *Simple* .The application uses the *Hadoop MapReduce* software and four Amazon services: *EC2*, *Simple DB*, *S3*, and *SQS*.

(a) The simplified workflow showing the two inputs, the regular expression and the input records generated by the web crawler; a third type of input are the user commands to report the current status and to terminate the processing.

(b) The detailed workflow; the system is based on message passing between several queues; four controller threads periodically poll their associated input queues, retrieve messages, and carry out the required actions. *DB* and executes the monitor task; it updates the status in the Amazon *Simple DB* domain, enqueues messages in the shutdown and the billing queues. The monitor task checks for the *Hadoop* status periodically, updates the *Simple DB* items with status/error and the Amazon *S3* output file. Finally, it deletes the message from the monitor queue when the processing is completed.

4. *The shutdown phase:* the shutdown controller thread retrieves the message from the shutdown queue and executes the shutdown task which updates the status and time stamps in the Amazon *Simple DB* domain; finally, it deletes the message from the shutdown queue after processing. The shutdown phase consists of the following steps:

(a) The shutdown task kills the *Hadoop* processes, terminates the *EC2* instances after getting *EC2* topology information from Amazon *Simple DB* and disposes of the infrastructure.

(b) The billing task gets the *EC2* topology information, *Simple DB* usage, *S3* file
and query input, calculates the charges, and passes the information to the billing service.

5. *The cleanup phase:* archives the *Simple DB* data with user info.

6. *User interactions with the system:* get the status and output results. The *GetStatus* is applied to the service endpoint to get the status of the overall system (all controllers and *Hadoop)* and download the filtered results from Amazon *S3* after completion. To optimize the end-to-end transfer rates in the *S3* storage system multiple files were bundled up and stored as *S3* objects; another performance optimization was to run a script and sort the keys, the URL pointers, and upload them in sorted order in *S3*. Also, multiple
fetch threads were started in order to fetch the objects.

This application illustrates the means to create an on-demand infrastructure and run it on a massively distributed system in a manner that allows it to run in parallel and scale up and down based on the number of users and the problem size.

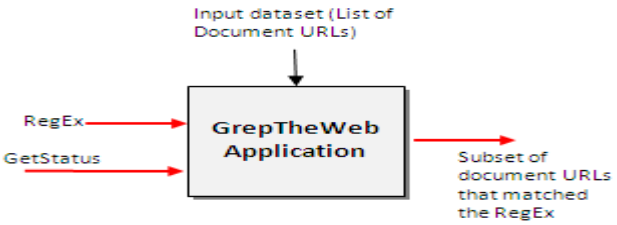

Fig. 4 : GrepTheWeb Architecture - Zoom Level 1

IJSER

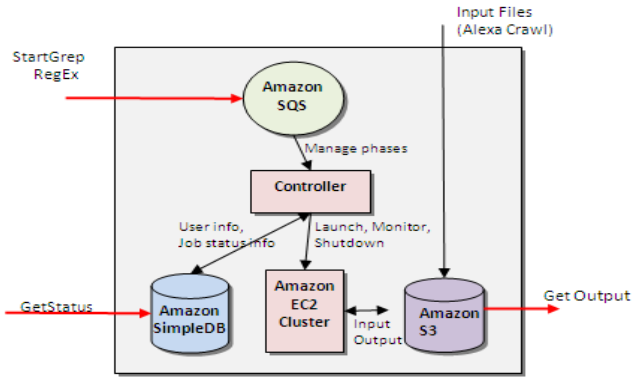

Fig. 5: GrepTheWeb Architecture - Zoom Level 2

GrepTheWeb architecture looks like as shown in Figure 5 (above). It uses the following AWS components:

Amazon S3 for retrieving input datasets and for storing the output dataset

Amazon SQS for durably buffering requests acting as a "glue" between controllers

Amazon SimpleDB for storing intermediate status, log, and for user data about tasks

Amazon EC2 for running a large distributed processing Hadoop cluster on-demand

Hadoop for distributed processing, automatic parallelization, and job scheduling

V. HADOOP MAP REDUCE

Hadoop is an open source distributed processing framework that allows computation of large datasets by splitting the dataset into manageable chunks, spreading it across a fleet of machines and managing the overall process by launching jobs, processing the job no matter where the data is physically located and, at the end, aggregating the job output into a final result.

It typically works in three phases. A map phase transforms the input into an intermediate representation of key value pairs, a combine phase (handled by Hadoop itself) combines and sorts by the keys and a reduce phase recombines the intermediate representation into the final output. Developers implement two interfaces, Mapper and Reducer, while Hadoop takes care of all the distributed processing (automatic parallelization, job scheduling, job monitoring, and result aggregation).

In Hadoop, there's a master process running on one node to oversee a pool of slave processes (also called workers) running on separate nodes. Hadoop splits the input into chunks. These chunks are assigned to slaves, each slave performs the map task (logic specified by user) on each pair found in the chunk and writes the results locally and informs the master of the completed status. Hadoop combines all the results and sorts the results by the keys. The master then assigns keys to the reducers. The reducer pulls the results using an iterator, runs the reduce task (logic specified by user), and sends the "final" output back to distributed file system.

Map Reduce Operation (in GrepTheWeb)

MAPPER: For each input record, extract a set of key/value pairs that we care about the each record

(LineNumber, s3pointer)->
(s3pointer, [matches])

REDUCER: For each extracted key/value pair, combine it with other values that share the same key

Identity Function

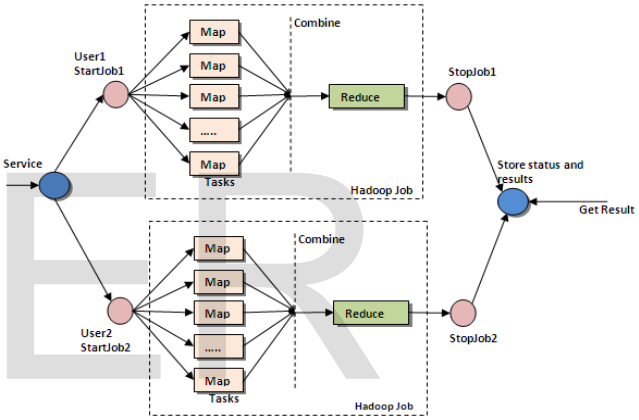

Fig 6: GrepTheWeb Hadoop Implementation

Hadoop suits well the GrepTheWeb application. As each grep task can be run in parallel independently of other grep tasks using the parallel approach embodied in Hadoop is a perfect fit.

For GrepTheWeb, the actual documents (the web) are crawled ahead of time and stored on Amazon S3. Each user starts a grep job by calling the StartGrep function at the service endpoint. When triggered, masters and slave nodes (Hadoop cluster) are started on Amazon EC2 instances. Hadoop splits the input (document with pointers to Amazon S3 objects) into multiple manageable chunks of 100 lines each and assign the chunk to a slave node to run the map task. The map task reads these lines and is responsible for fetching the files from Amazon S3, running the regular expression on them and writing the results locally. If there is no match, there is no output. The map tasks then passes the results to the reduce phase which is an identity function (pass through) to aggregate all the outputs. The "final" output is written back to Amazon S3.

Mapper Implementation-

IJSER

Key = line number and value = line in the input dataset
i)Create a signed URL (using Amazon AWS credentials) using the contents of key-value
ii) Read (fetch) Amazon S3 Object (file) into a buffer
iii) Run regular expression on that buffer
iv)If there is match, collect the output in new set of key-value pairs (key = line, value = up to 5 matches)

Reducer Implementation - Pass-through (Built-in Identity Function) and write the results back to S3.

VI. CONCLUSION

Cloud technologies work well for most pleasingly-parallel problems. Their support for handling large data sets, the concept of moving computation to data, and the better quality of services provided such as fault tolerance and monitoring, simplify the implementation details of such problems over the traditional system. Most cloud technologies support the concept of moving computation to data where the parallel tasks access data stored in local disks.. Instead of building your applications on fixed and rigid infrastructures, Cloud Architectures provide a new way to build applications on on-demand infrastructures. GrepTheWeb demonstrates how such applications can be built.

VII. REFERENCES

[1] P. S. Pacheco, *Parallel programming with MPI*. San Francisco, CA,USA: Morgan Kaufmann Publishers Inc., 1996.
[2] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1 –10.
[3] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1 –10.
[4] L. Yu and et al., "Harnessing parallelism in multicore clusters with the all-pairs, wavefront, and makeflow abstractions," *Journal of Cluster Computing*, vol. 13, no. 3, pp. 243–256, 2010.
[5]"The directed acyclic graph manager," http://www.cs.wisc.edu/condor/dagman, 2002.
[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Operating Systems Design and Implementation*, 2004.
[7] Y. Sugita and Y. Okamoto, "Replica-exchange molecular dynamics method for protein folding," *Chemical Physics Letters*, vol. 314, no 1-2, pp. 141 – 151, 1999.
[8] P. Brenner, C. R. Sweet, D. VonHandorf, and J. A. Izaguirre, "Accelerating the replica exchange method through an efficient all-pairs exchange," *Journal of Chemical Physics*, vol. 126, p. 074103, February 2007.
[9] K. Al-Tawil and C. A. Moritz, "Performance modeling and evaluation of mpi," *Journal of Parallel and Distributed Computing*, vol. 61, no. 2, pp. 202 – 223, 2001.
[10] M. Resch, H. Berger, and T. B¨onisch, "A comparison of mpi performance on different mpps," in *Proceedings of the 4th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer-Verlag, 1997, pp. 25–32.
[11] T. Matthey and et al., "Protomol, an object-oriented framework for prototyping novel algorithms for molecular dynamics," *ACM Transactions on Mathematical Software*, vol. 30, pp. 237–265, September 2004.
[12] J. Phillips, G. Zheng, S. Kumar, and L. Kale, "Namd: Biomolecular simulation on thousands of processors," in *Supercomputing, ACM/IEEE 2002 Conference*, November 2002.
[13] E. Lindahl, B. Hess, and D. van der Spoel, "Gromacs 3.0: a package for molecular simulation and trajectory analysis," *Journal of Molecular Modeling*, vol. 7, pp. 306–317, 2001.
[14] W. Gentzsch, "Sun grid engine: Towards creating a compute power grid," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, ser. CCGRID '01, 2001.
[15] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM - 50th anniversary issue: 1958-2008*, vol. 51, no. 1, 2008.
[16] XIAO Yonghao, HUANG Qingnan (2011). Parallel Computation of Impact Problems Based on Block Data Structure (in Chinese). *Computer Aided Engineering, 1*, 33-36.
[17] Calvin Lin & Lawrence Snyder (2009). *Parallel Program Design Principles*. Beijing: Mechanical Industry Press.
[18] ZHU Yongzhi, LI Bingfeng, SUN Tingting & LI Pei (2011). Research on Scalability of Parallel Computing System (in Chinese). *Computer Engineering and Applications*.
[19] Amazon.com, "Elastic compute cloud (ec2)," http://www.aws.amazon.com/ec2.
[20] Jaliya Ekanayake, Xiaohong Qiu, Thilina Gunarathne, Scott Beason, Geoffrey Fox High Performance Parallel Computing with Clouds and Cloud Technologies to appear as a book chapter to Cloud Computing and Software Services: Theory and Techniques, CRC Press (Taylor and Francis), ISBN-10: 1439803153. http://grids.ucs.indiana.edu/ptliupages/publications/cloud_handbook_final-with-diagrams.pdf
[21] Open source MapReduce Apche Hadoop, http://hadoop.apache.org/core/

IJSER