Volume 6, Issue 5, May 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Autonomic Resource Allocation and Delay Optimization of Scientific Computing on Cloud

Srutica Tyagi, Sachin Chaudhary
CSE & U.P. Technical University,
Uttar Pradesh, India

Abstract- Cloud computing is the latest technology in computing service where computing is leased out as a service on a pay-per-use basis. This has a huge potential in providing scientific computing to those users who do not have access to traditional resources like grids, clusters etc. Cloud Computing helps researcher's process complex workloads by providing the cost effective, scalable and secure compute, storage and database capabilities needed to accelerate time-to-science. These resources can be released when they are no more needed. Such services are often offered within the context of a Service Level Agreement (SLA), which ensure the desired Quality of Service (QoS). In this paper the cloud framework Eucalyptus has been studied and is used for the purpose of scientific computing. The chosen problem - Matrix Multiplication is set up using the Message Passing Interface, a leading standard for message passing libraries. The experiments are done on and the resulting response times are observed. These response times are modeled as functions of the size of the jobs and the resources allocated. Using these models, a resource allocation algorithm has been proposed. The algorithm is based on the optimization problems that reduce the overall delay over the deadline for the batch of jobs submitted every triggering interval while maximizing resource utilization. The proposed algorithm has been simulated and shown to be superior to a simple sharing algorithm in terms of delay minimization.

Keywords: Scientific computing, computational science, Cloud computing, Eucalyptus, high-performance computing.

I. INTRODUCTION

Computing in the cloud lets the user purchase computing resources as a metered service over a network (typically the Internet) and not as a product. This has been made possible due to the advances in Operating System Virtualization and the Internet. Cloud computing allows almost any server environment to be replicated and scaled instantly. Several companies found Cloud Computing much more economical than setting up the hardware infrastructure on their own. Cloud computing [2] is a style of computing in which, typically, real-time scalable resources are provided "as a service (aaS)" over the Internet to users who need not have knowledge of, expertise in, or control over the cloud infrastructure that supports them. The provisioning of cloud services can be at the Infrastructural level (IaaS) or Platform level (PaaS) or at the Software level (SaaS). A cloud computing platform dynamically provisions, configures, reconfigures, and de-provisions servers as requested. This ensures elasticity of the systems deployed in the cloud. Among the three standard models of deployment of Cloud Gibson et al. (2012), SaaS is the most popular and widely used model of Cloud computing. SaaS uses the Internet to deploy applications on the Cloud that are managed by third party vendors He (2010). Users access the application via the web browser on their side. This eliminates the need to download and install software to run these. Scientific computing is one such SaaS application that can be delivered over the Cloud Yang et al. (2011); He et al. (2010). Scientific computing involves the construction of mathematical models and numerical solution techniques to solve scientific, social scientific and engineering problems on distributed systems Jakovits and Srirama (2013). Grid Computing gained high popularity in the field of scientific computing through the distributed resource sharing models deployed in academic institutions. Scientific computing is a high-utilization workload requiring huge number of computing resources traditionally employed on Grids.

Ostermann et al. (2009) tells us how Cloud Computing can be an attractive alter-native to Grid Computing for scientific computing scientists. Since a cloud promotes leasing resources than setting up one's own infrastructure, this is an economic alternative for academic institutions. This also eliminates the burden of constantly updating the hardware due to advances in technology. The cloud eliminates the overhead costs arising due to over-provisioning of occasionally needed resources. The cloud can scale up resources in a fast and cost effective fashion. Despite these scientific computing on the Cloud also presents some challenges like slower interconnects, the pricing model, data management, the resource allocation model, security, etc Kaur and Chana (2010).

The contribution of this work is to be able to provide parallelizable scientific computing applications on a Cloud environment. The setting up of the Cloud environment Eucalyptus has been studied in detail and implemented. The

problems frequently encountered in scientific computing have been chosen and a parallelized algorithm has been implemented on the cloud environment. Message Passing Interface, a leading standard for message passing libraries has been studied and has been implemented for parallelizing the jobs. The response times of these jobs are modeled as functions of the size of the jobs and the resources allocated. Furthermore a resource allocation algorithm has been proposed that seeks to reduce the overall delay over the deadline for the jobs.

II. LITERATURE REVIEW

A. Scientific Computing on the Cloud - A Distributed Computing Approach

Scientific Computing is one of the leading disciplines in information technology with varied application in fields such as economics, science and engineering. It is the practice of aggregating the computing resources in such .a way that it delivers much higher performance than computations on a personal desktop or workstation. Due to specific performance requirements, it is common to operate high performance computing re-source in private and thus the access to these are often restricted. Also jobs have to wait in a queue for resource allocation and execution. Also it may happen that these physical machines are underutilized because of the fluctuating demands within the particular organization

1) Advantages

Thus HPC or scientific computing on the cloud can be alternate solution to the computing needs of the organization. A cloud computing approach can be both cost effective and efficient alternative to traditional HPC approaches. There are several advantages to using a cloud computing approach:

1. Large providers can set up the required infrastructure bring down the overall running cost and thus computing resources can be made available at lower costs.
2. The use of VMs can allow users to secure administrative privileges and thus customize usage according to their requirements, from choosing operating systems down to the various libraries.
3. The continuous availability and scalability of cloud ensure virtually infinite pool of resources to choose from at any point of time. Thus jobs need not wait in queue for resources.
4. Clouds can provide isolation of multiple workloads on the same physical re-sources and networking infrastructure.
5. Dynamic provisioning ensures that the computing resources can be scaled up and down according to the users' workload demand fluctuation.
6. Service Level Agreements can guarantee network performances and other Quality of Service (QoS) constraints.
7. Live Migration of VMs is a concept that allows seamless transfer of VMs over physical resources during operations. The advantage is that maintenance work can be run on the physical resources without interrupting the jobs or processes.

B. Distributed Computing - Definition and Frameworks

Distributed computing refers to the parallelization of a large computational job into several smaller computational tasks and executing these tasks on different nodes. Nodes are autonomous computational resources with its own local memory that communicate with each other by passing messages on the network. First a MATLAB Distributing Computing Server was considered but was found to be inadequate for the work due to system constraints. Then a Python based approach was considered. For communication purposes the two most prevalent approaches have been MPI and MapReduce. Finally MPI was chosen for the simplicity and flexibility provided.

C. Message Passing Interface

The Message Passing Interface is a standardized and portable message passing system designed to function on a wide variety of parallel systems. The standard itself is not a library, but defines the syntax and semantics of the library routines for a language independent communications protocol. MPI primary addresses the message passing parallel programming model in which data is moved across the address spaces of processes through cooperative operations. Since the release of MPI, it has become the leading standard for message passing libraries for parallel systems and has achieved widespread implementation.

1) Functionality of MPI

MPI interface is meant to provide essential virtual topology, synchronization and communication functionality between a set of processes that have been mapped to nodes/instances. Typically each core in a multicore machine will be assigned a single process. This assignment happens at runtime through the agent that starts the MPI routine. Functionalities include point-to-point rendezvous type send/receive operations, choosing between Cartesian or graph-like logical process topology, exchanging data between process pairs, combining the partial results of computations (gather and reduce), synchronization of nodes. Point-to-point communication can be of synchronous, asynchronous or buffered type as well as blocking/non-blocking type. Collective communications like broadcast and scatter are also supported. Latest implementations of MPI also support dynamic process management allowing addition of new processes during program execution.

2) Programming Model of MPI

The MPI was first developed as a communication protocol for distributed systems. As developments were made in architectural trends, shared memory systems were also combined into MPI creating hybrid distributed and shared memory systems. The libraries were adapted to handle both types of memory architectures seamlessly. In most MPI

implementation, a fixed set of processes are created and each process is assigned to a processor. However each process may execute a different program on a different set of data. Hence MPI model conforms to the 'multiple program multiple data' (MPMD) model.

3) Advantages

1. Standardization: MPI is the only message passing library that can be considered a standard. It is supported on all types of HPC platforms.
2. Portability: The user can seamlessly port the applications across platforms (that are MPI compliant) by making minimal changes to source code.
3. Functionality and Performance Opportunities: There are over 440 routines de-fined in the latest MPI release. Also vendor specific implementations can exploit hardware features for maximal performance.
4. Availability of rich documentation and support.

4) Communicators and Groups

A group is an ordered set of processes. Each process in a group is associated with a unique integer rank. Rank values range from 0 to N-1, where N is the number of processes in the group. A process can belong to multiple groups. Group is a dynamic object in MPI and can be created or destroyed during the execution of the program. For any message being passed, the source and destination are identified by the process rank of the process within that group. The communicator is an object that defines the 'communication universe' within the MPI framework. It is a logical unit that defines which processes are allowed to send and receive messages. Intracommunicator is the communicator that is used for communication within a single group of processes. Intercommunicator is the communicator that is used for communication across different non overlapping groups. Communicators are also dynamic, i.e. they can be created or destroyed during the execution of the program. Each communicator gives each contained process an independent identifier and arranges its contained processes in an ordered topology.

D. Python

Python is a widely used general-purpose, high-level programming language. It has an efficient high-level data structures and a simple but effective approach to object-oriented programming with dynamic typing and dynamic binding. It has a holistic language design with emphasis on readability and concise coding. It has the perfect balance of high level and low level programming. It is an open source programming languages with an impressive standard library and external libraries being developed by the enthusiastic Python community. Python has good language interoperability. Python has an impressive support for scientific computing. SciPy is a computing environment and open source ecosystem of software for Python programming language that is used by scientists, analysts and engineers doing scientific computing and technical computing. SciPy also refers to the open source Python library of algorithms and mathematical tools that are at the crux of the SciPy environment.

1) MPI for Python

There are several Python packages available that wrap MPI as a library and allow MPI functions to be called from Python. These include pyMPI, maroonmpi, mpi4py, myMPI, Pypar etc. The mpi4py library Dalcin (2012) provides an interface very similar to the MPI-2 standard C++ Interface. It supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communications of Python objects, as well as optimized communications of certain Python object like NumPy arrays. We have chosen mpi4py for our implementation.

E. Problem Formulation

1) Application Parallelization and Modeling

For the experiments we chose the Dense Matrix Multiplication Problem, a commonly resource intensive problem. The main motivation was to be able to setup parallelized application, so that the user can run the algorithm on their datasets within the time constraints.

Dense Matrix Multiplication Problem

Matrix multiplication, from an academic viewpoint has been one of the first and most widely studied application of parallelization. Its importance as a problem to be solved and relevance in several fields has pushed research constantly. Some specific characteristics of this problem with regards to its design and implementation of a parallel algorithm are as follows.

1. Computational Independence: Each element of the output matrix is independent from all the other elements. This allows for a wide flexibility in terms of parallelization.
2. Data Independence: In the case of dense matrix multiplication, the number and type of operations to be run are independent of the data.
3. Data Organization: Since the data is organized in two-dimensional structures, it allows for flexibility in algorithm as well as creating suitable topologies of processes for efficient performance.

F. Problem Description

Given two square matrices $A_{n\,n}$ and $B_{n\,n}$ of dimension n where each of its elements are denoted as a_{ij} and b_{ij} with 1 i; j n, the matrix C resulting from the operation of multiplication of matrices A and B, C = A B, is such that each of its elements

$c_{ij} = a_{ij}b_{ij}$ with 1 i; j n.

1) Sequential Approach - Naive Matrix Multiplication Algorithm

The algorithm for the naive matrix multiplication is given in Algorithm 1.1.

Algorithm 1.1 Naive Matrix Multiplication

Require: Matrices $A_{n\ n}$ and $B_{n\ n}$
Ensure: Matrix $C_{n\ n} = A_{n\ n}\ B_{n\ n}$
for i=0 to n
for j=0 to n do
for k = 0 to n do
$c_{ij} = c_{ij} + a_{ik*}b_{kj}$

The sequential algorithm has a complexity of $O(n^3)$

2) Distributed Approach - Block Strip Multiplication Algorithm

In this distributed approach only one of the two matrices are broadcast to all the processes once and the other one is stripped into blocks. There is only one iteration. The algorithm for the block strip multiplication algorithm run on p different processes is given in Algorithm 2.2.

Algorithm 2.2 Block Strip Matrix Multiplication

Require: Matrices $A_{n\ n}$ and $B_{n\ n}$
Ensure: Matrix $C_{n\ n} = A_{n\ n}\ B_{n\ n}$
if rank =0 then //Load Balancer
Divide(B; k) // Divides matrix B into p vertical blocks (B_1; B_2; : : : : : : ; B_p)
Broadcast(A)
for i=0 to p do
Send(B_i; rank = i)
Receive(C_i)
C = Aggregate(C_i)
else // Processors
Receive(A)
Receive(B_i)
$C_i = A\ B_i$
Send(C_i; rank = 0)

3) Experiment

The experiment was done using randomly generated dense matrices of sizes varying from 1000*1000 to 3000*3000.Each run of the matrix multiplication was distributed over a set of VMs. The numbers of VMs were varied from 1 to 10 VMs for each matrix multiplication. It is assumed that no other job is running on the VMs and that there is no waiting time for any job. The response times of each job were measured.

III. PROPOSED WORK

A. Virtual Machine Allocation - Delay Optimization and Algorithm

Optimization Problem Formulation

In our analysis we consider the cloud model wherein parallelizable applications like the one described in the previous chapter arrive at the loadbalancer. The loadbalancer creates batches of several jobs and allocates the resources at regular intervals. At every resource allocation trigger, the loadbalancer computes the number of resources to be provided based upon the algorithm proposed.

Consider the batches as a function of time as Batch(t). These are considered at every resource allocation trigger to be composed of several jobs
(A_1(t); A_2(t); : : : ; A_m(t)). Each of these jobs specifies the following parameters:

1. Job Size (size): This is the parameter that decides the size of the job. Matrix dimensions in the case of Matrix Multiplication.
2. Due Time (d): This is a parameter that specifies the due time expected by the user. It is decide based on the priority of the job assigned by the user.

Let (x_1; x_2;::: ; x_m) refer to the number of VMs to be assigned for each of the jobs. Then the expected worst case running time of each of the jobs can be computed according to the models described in the previous section. Let us denote the expected worst case run time for job A_i with size s_i and due time d_i run on j VMs be given as $t_i(j) = F(s_i; j)$. Our aim is to minimize the total delay over the expected due time experienced by the user. Hence the cost function to be minimized is given as $^{m}P_{ij}$ where $_{ij}$ represents the excess delay over the expected due time when run on j resources and is given as

ij= 8 $t_i(j)\ d_i$: $t_i(j)>d_i$
< 0 : ti(j) < di

At every resource allocation trigger the loadbalancer runs a check on the current utilization of the system and returns the number of free resources (VMs) that can be allocated.

B. Proposed Algorithm

The jobs arriving are grouped into batches at regular trigger intervals as shown in Fig. 1

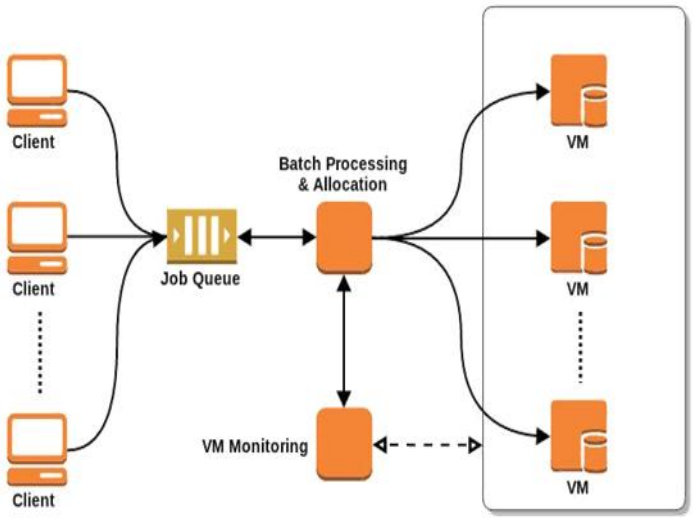

Figure 1: Job Queuing and Allocation

At each trigger instant, we have a batch of m jobs and n resources need to allocated to minimize the overall ${}^{m}P_{ij}$. We create a matrix C where the row corresponds to each job and the columns correspond to the number of VMs allocated. The proposed algorithm first computes the total number of resources required for the jobs n_{init} for minimizing the delay without imposing the constraint on total number of available resources. Then it iterates from n_{init} to the actual value n. In each iteration it looks for the jobs that will enter the least amount of penalty to the cost function. The detailed algorithm is given as

Algorithm 3.1 Resource Allocation Algorithm

Require: Model equations, n number of VMs, Job numbers and details

Ensure: x_i number of virtual machines to be allocated for each job i

for i = 1 to m do
Compute j_i such that $t_{ij} = d_i$
$J_i = dj_ie$
ij = tiJi di
if (Ji) n then
Allocate J_i VMs to job i
else
k = 0
for i = 1 to m do
if $J_i = 1$ then
Allocate $J_i = 1$ VM to job i
extra_delay$_i$ = 0
k = k + 1
else
extra_delay$_i$ = iJi 1 iJi
m = m k
n = n k
iter = (Ji) n
while P == 0 do
for $J_i > 1$ do
I = arg min$_i$ extra_delay$_i$
J_i = Ji 1
if $J_I > 1$ then
extra_delay$_i$ = iJi 1 iJi
else
extra_delay$_i$ = 0
iter = iter 1
Allocate J_i VMs to job I
m
Total Excess Delay is x_i where x_i = iJi
i=1

C. Simulation and Results

As proof of concept, we have simulated the proposed algorithm and compared it with a simple shared allocation algorithm. In the simple shared allocation algorithm, we divide the total resources available equally among the jobs in the

batch. . We simulate the two algorithms using the multiprocessing package from python. This package lets us spawn multiple processes that can work on a queue structure. We create 3 processes for simulating each of the algorithms. The first process generates batches of jobs at regular interval and pushes it into the queue. The second process pops these batches from the queue, executes the algorithms, starts the jobs on the simulated VMs and computes the delay. The third process monitors and updates the execution of the jobs on the simulated VMs with time.

The jobs were sent in the form of 20 batches at regular intervals of 5 seconds. The no of virtual machines initially allotted were 5. They were scaled up to 10 whenever the no of jobs were more than available free VMs. It is evident that the proposed minimum delay algorithm is superior to a simple shared algorithm. Also a comparison of the VMs utilization reveals that the proposed algorithm completes the jobs faster than the shared algorithm

IV. CONCLUSION

In this work we have successfully set up the Eucalyptus Cloud framework on the local physical systems to function as a Private Cloud. Using the Message Passing Interface library, we were able to deploy the parallel Dense Matrix Multiplication application in the distributed environment to reduce time. This application can thus be deployed over the cloud for use by the users in the scientific community who would like to take advantage of the benefits offered by clouds. Thus the user can get results without bothering about the underlying resource management or implementation. The profiles of the response times for each of the applications have been modeled as a function of the job sizes and the resources allocated. Based on these benchmarking tests, a new resource allocation algorithm has been proposed to reduce overall delay over expected due time for jobs in a batch-wise fashion. The proposed algorithm has been simulated and shown to be superior to a simple sharing algorithm in terms of delay minimization.

REFERENCES

[1] Buyya, R., C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic (2009). Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6), 599 – 616. ISSN 0167-739X. URL http://www.sciencedirect.com/science/article/pii/S0167739X08001957.

[2] Dalcin, L. (2012). MPI for Python, Release 1.3. URL http://mpi4py.scipy. org//.

[3] Eucalyptus Systems, I. (a). Eucalyptus 3.1.2 Installation Guide. URL https:// www.eucalyptus.com/.

[4] Eucalyptus Systems, I. (b). Eucalyptus 3.1.2 User Guide. URL https://www. eucalyptus.com/.

[5] Gibson, J., R. Rondeau, D. Eveleigh, and Q. Tan, Benefits and challenges of three cloud computing service models. In Computational Aspects of Social Networks (CA-SoN), 2012 Fourth International Conference on. 2012.

[6] Gong, C., J. Liu, Q. Zhang, H. Chen, and Z. Gong, The characteristics of cloud computing. In Parallel Processing Workshops (ICPPW), 2010 39th International Conference on. 2010. ISSN 1530-2016.

[7] He, H., Applications deployment on the saas platform. In Pervasive Computing and Applications (ICPCA), 2010 5th International Conference on. 2010.

[8] He, Q., S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, Case study for running hpc applications in public clouds. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-942-8. URL http://doi.acm.org/10.1145/ 1851476.1851535.

[9] Jakovits, P. and S. Srirama, Adapting scientific applications to cloud by using distributed computing frameworks. In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on. 2013.

[10] Kaur, P. and I. Chana, Unfolding the distributed computing paradigms. In Advances in Computer Engineering (ACE), 2010 International Conference on. 2010.

[11] Ostermann, S., R. Prodan, and T. Fahringer, Extending grids with cloud resource management for scientific computing. In Grid Computing, 2009 10th IEEE/ACM Inter-national Conference on. 2009.

[12] Yang, G., Z. Zhu, and F. Zhuo, The application of saas-based cloud computing in the university research and teaching platform. In Intelligence Science and Information Engineering (ISIE), 2011 International Conference.

