BSPCloud: A Hybrid Distributed-memory and Shared-memory Programming Model

Xiaodong Liu[1], Weiqin Tong[1], Fu ZhiRen[2] and Liao WenZhao[2]

[1]*School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China*
[2]*China Telecom Corporation Ltd. Shanghai Branch, Shanghai, 200085, China*
liuxiaodongxht@qq.com, wqtong@mail.shu.edu.cn, fuzr@shtel.com.cn, liaowenzhao@189.cn

Abstract

Programming models for cloud computing has been a research focus recently. Current programming models for cloud computing mainly focus on improving the efficiency of the cloud computing platforms but little has been done on the performance predictability of models. This paper presents a cloud computing programming model, called BSPCloud. The BSPCloud has the advantages of predictable performance. BSPCloud uses a hybrid of distributed-memory and shared-memory bulk synchronous parallel (BSP) programming model. Computing tasks are first divided into a set of coarse granularity bulks which are computed by the distributed-memory BSP model, and each coarse granularity bulk is further divided into a set of bulk threads which are computed by the shared-memory BSP model. BSPCloud makes full use of the multi-core architecture, and more importantly, the performance of the BSPCloud is predictable. We have implemented a proof-of-concept BSPCloud parallel programming library in java. We give the implementation of BSPCloud parallel programming library and show how the library implements hybrid programming. We detail applying the BSPCloud library on matrix multiplication. The performance predictability and speedup are evaluated in the cloud platform. We give the results of experiments.

Keywords: *Cloud programming, Programming model, BSPCloud, Bulk synchronous parallel*

1. Introduction

Increasing Internet business motivate server consolidation in data centers, which has become the foundation of cloud computing. Through virtualization-based cloud computing, multiple computers allow running as virtual machines (VM) in a single physical computer. With the ability to reduce hardware resources and provide on-demand service, the cloud computing has become very popular. The cloud computing platform integrates vast computing network and storage resources together and provides service on demand through network. Users can apply cloud resources on demand and pay for it by hours.

The development and deployment become convenient through the cloud computing platform. Meanwhile, cloud computing brings new challenges to programming model for cloud computing. Traditional programming models cannot adapt to cloud computing very well. Firstly, the cloud computing center is usually composed of thousands of computers and it provides service for common users. The programming model for cloud computing should easily programming and deadlock avoidance. Secondly, in cloud computing environment, multiple virtual machines (VMs) are running in a single physical computer. Computing tasks are performed in virtual machines will contend I/O resource with others in the same physical computer. How to avoid the I/O resource competition and reduce the I/O overhead is a big problem of the programming mode for cloud computing.

However, some progresses have been made on the cloud computing programming model,

such as Google's MapReduce [1] and Microsoft's Dryad [2]. MapReduce divides the computing process into two stages: map and reduce. In map phase, the task is divided into a set of sub-tasks which are executed in parallel. In reduce phase, the results of sub-tasks' are merged together. A Dryad job is expressed as a directed acyclic graph where each vertex is a program and edges represent data channels. Concurrency arises from Dryad scheduling vertexes to run in parallel on a set of computers. Dryad is mainly for coarse-grain data parallel. There has also been some work based on these two models [3-5]. However, this work has largely focused on program parallelism. In cloud computing environment, users use cloud resources by payment. Performance predictability of the programming model can guide users to apply and use cloud resources reasonably. Therefore, it is very important for the programmer to rely on a simple yet realistic cost model when one programming a cloud computing application. Thus research on programming models whose performance can be predicted is of great significance.

The Bulk Synchronized Parallel (BSP) [6] model has the advantages of performance predictability, easily programming and deadlock avoidance. Because of these advantages, the BSP model has been used in many programming environment. The papers [7, 8] extend the BSP model to solve programming on heterogeneous grid environment, the paper [9] applies BSP model in parallel database and the paper [10] applies it in research engine. However, BSP model cannot be used to cloud computing directly. Firstly, the original BSP library is used in cluster which is composed of single core nodes. Nowadays, the multi-core processor is very ubiquitous. The nodes of cloud platform are almost all multi-core computers. Secondly, with the increasing maturity of virtualization technology, virtual machine systems have been widely used. A virtual machine can have many virtual CPUs, and hundreds of virtual machines can run in a physical machine. In cloud computing environment, the cloud platform provides users with virtual machines. In this paper, we adapt the BSP model to cloud computing environment and propose a cloud computing programming model—BSPCloud.

The BSPCloud is a hybrid of distributed-memory and shared-memory programming model. Computing tasks are first divided into a set of coarse granularity bulks, which are computed by the distributed-memory BSP model. In every VM, the coarse granularity bulk is further divided into a set of bulk threads, which are computed by the shared-memory BSP model. The advantage of the hierarchical programming model is that it can make full use of multi-core architecture.

The proposed programming model is implemented in java. We show that the BSPCloud library can attain proper speedups on cloud computing environments. More importantly, the BSPCloud has good performance predictability.

Our major contributions are the following aspects:

- We propose a programming model for cloud computing, called BSPCloud. The BSPCloud uses a hybrid distributed-memory and shared-memory BSP model, which can make full use of multi-core cluster architecture.
- A proof-of-concept of BSPCloud library has been implemented in java. We show BSPCloud attains proper speedups and has good performance predictability.
- We show shared-memory BSP can be directly used in the multi-core computer.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 describes the BSP programming model and explains why hybrid method is adopted in BSPCloud. Section 4 presents the BSPCloud programming model framework. Section 5 gives the BSPCloud programming library and its implementation. Section 6 gives the experiment results. Section 7 concludes our work.

2. Related Work

We divide previous work into two groups: the cloud computing programming model and the BSP programming model.

2.1 Cloud Computing Programming Model

Google's MapReduce [1] is a programming model and an associated implementation for processing massive data. MapReduce hides the details of parallelization、fault tolerance and load balancing. Users only need to implement two functions: Map and reduce, MapReduce can automatically parallelize the computation across thousand of machines. The MapReduce has its own limitations. Its one input, two-stage data flow is a little rigid and it is difficult to reuse and maintain. Hadoop [11] is the open-source implements of MapReduce.

Microsoft's Dryad [2] is more flexible than MapReduce. A Dryad job is a directed acyclic graph where each vertex is a program and edges represent data channels. Concurrency arises from Dryad scheduling vertices to run simultaneously on clusters of machines. Dryad allows arbitrary numbers of input and output, and it is mainly used in coarse-grain data-parallel applications.

Yahoo's Pig Latin [3] and HadoopDB [4] combine high level declarative style of SQL and low-level procedural style of map-reduce which are specialized in massive data process.

2.2 BSP Programming Model

The Google's Pregel [12] is a distributed parallel programming model which based on BSP model. Pregel is used to process large graphs. The programs are expressed as a set of super –steps, in each of which a vertex can receive messages sent in the previous super-step, and sends messages to other vertices, and modifies its own state and mutates graph topology. Pregel is flexible enough to express a broad set of algorithms. However, Pregel is designed for sparse graphs at present. Dense graphs and dense communication computation cannot solve still. Hama [13] is a BSP computing framework on top of Hadoop Distributed File System (HDFS) for massive scientific computations such as matrix, graph and network algorithms. Ham provides compatibility with Hadoop, scalability for the large problem size, flexibility with the plug-in engineer interface. However, Hama provides BSP library base on Hadoop. The storage style of Hadoop is not very suitable for BSP model.

Multi-BSP [14] is a multi-level model that has explicit parameters for processor numbers, memory/cache sizes, communication costs, and synchronization costs, which aimed at capturing the most basic resource parameters of multi-core architectures. A MulticoreBSP library which targets share-memory multi-core system is introduced in [15].

3. BSP Model

The Bulk Synchronous Parallel (BSP) [4] model is first proposed by Harvard's valiant which used to bridge parallel architecture and programming language. A BSP model has three independent and interdependent components: (a) a set of processor/memory pairs, (b) a network providing point-to-point communication, (c) synchronization barrier that facility for synchronizing all or a subset of the components.

A computation of BSP model consists of a sequence of super-steps. The execution of each super-step is divided into three ordered phases.

- Each processor computation locally using only its local data.
- Communication occurs among processor, processor send data to others in this phase.
- Barrier synchronization which ensures the data has been sent to destination and can be use in the next super-step.

One of the most important advantages of BSP model is that its performance can predictable. The cost model of BSP Depend on three parameters: p, g and l. The number of processor/memory pairs is given by p, and g is defined as network throughput rate, and l is the global synchronization time. The time of each super-step can be expressed by the following formula:

$$T = \sum_{s=1}^{S} \max_{1 \le i \le p} \omega_i^s + \sum_{s=1}^{S} \max_{1 \le i \le p} h_i^s \times g + l \times s \tag{3.1}$$

Where the computing time of processor i is given by w_i, h_i is the data needed to be communicated and s is the total number of super-steps.

The original BSP Library such as BSPlib is used in cluster which is composed of single core nodes. However, the BSP model cannot be used in cloud computing directly. In cloud computing environment, the physical machines are almost all multi-core processor. These physical machines are not provided to users directly. The machines which users apply are virtual machines. A virtual machine can has many virtual CPUs (VCPUs), and hundreds of virtual machines can run in a physical machine. The programming model for cloud computing should make full use of the multi-core or multi-VCPU architecture of the VM and reduce resources competition with other VMs located in the same physical machine.

Our original research maps each bulk to a VCPU. We find that the computing resource is made full use of, but the communication phase and synchronization phase waste much time. This lead the performance of BSP model is very bad. This is caused by the following reasons:

Firstly, multiple bulks in the same VM will contend I/O resource in the communication phase. There is only one communication channel can existence, when one bulks send data, the others in the same VM will have to wait.

Secondly, the synchronized time extends with the number of bulks increase. Too many bulks will waste much synchronized time.

In order to improve the performance, in our BSPCloud, we use a hybrid of distributed-memory BSP and shared-memory BSP system. Each VM acts as a bulk, and they communicate through network. VMs are used to compute coarse granularity tasks. Each VM further divided tasks into finer granularity tasks to exploit share-memory computation. In each VM, multiple threads are used to compute finer granularity tasks communication occurred in memory, which can avoid contend the I/O resource. Using the hybrid model, the synchronization phase will use a hierarchical approach, which will reduce synchronized time. Because of these advantages, the BSPCloud has a good speedup and scalability.

4. BSPCloud Framework

BSPCloud is a parallel programming model for cloud computing which combine distributed-memory BSP and shared-memory BSP. In this section, we describe BSPCloud architecture and its model.

4.1 BSPCloud Architecture

The BSPCloud model uses hierarchical control and communication mechanism. As is shown in figure 1, BSPCloud is composed of a BspJobTracker and a set of BulkTrackers. The BspJobTracker has two components: schedule and control. Schedule is responsible for deciding when the BspJob is running. Control is responsible for splitting tasks and controlling it run. When the schedule selects a BspJob to run, control assigns it to a set of BulkTrackers and controls their running. The BulkTracker also has two components: control and monitor. Control assigns the BulkTracker's tasks into a set of bulk threads and monitor is used to monitor the status of the bulk thread.

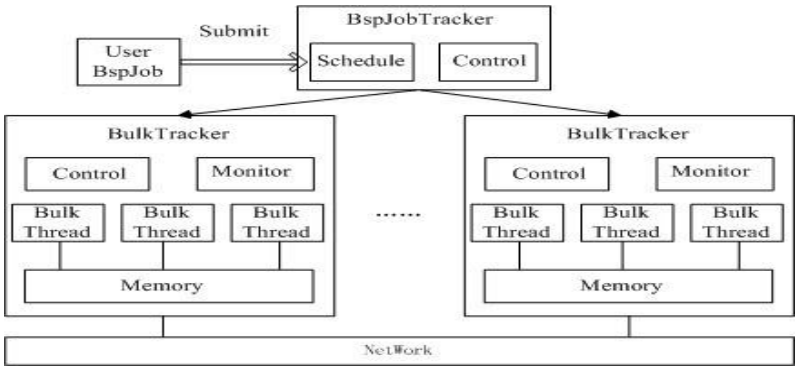


Figure 1. BSPCloud System Architecture

4.2 BSPCloud Communication Model

BSPCloud use hierarchical communicate model. A bulk thread can only communicate with the others located in the same VM. The communication between bulk threads uses shared-memory mechanism. When one bulk thread needs to communicate with others, it put the data into the memory. The others can get the date in the next super-step. The communication between VMs uses distributed-memory mechanism. Three is a communication channel between any two BulkTrackers. A VM can communicate with others through the BulkTrackers' communication channel. Hierarchical communicate model can save a lot of communication consumption, which can enhance speedup and scalability of BSPCloud.

4.3 BSPCloud Synchronization Model

The synchronization mechanism of distributed-memory and shared-memory uses the same principle. There is a bulk acted as master bulk. In the synchronization phase, all bulks send synchronization message to the master bulk and wait for response from the master bulk. When master bulk receives all synchronize message, it broadcasts next super-step message to all bulks. Once a bulk has received response message, it can enter the next super-step.

5. BSPCloud Programming Library and Implementation

We have implemented BSPCloud in java. This section describes the BSPCloud programming library and its implementation.

5.1 Define a BSPCloud Program

A generic program of distributed-memory BSP model is defined to be a class BspCloud. The class BspCloud includes two abstract functions bspOperate() and bspDataMap(), it also includes the basic functions of distributed-memory BSP model. A genetic program of shared-memory BSP model is defined to be a class BulkThread. The class BulkThread includes an abstract function parallelComp() and the basic functions of shared-memory BSP model. Table 1 shows the primary functions of distributed-memory and shared-memory BSP model.

Table 1. The Primary Functions Provide by Distributed-memory and Shared-Memory BSP Model

Distributed-memory model	Shared-memory model
getBspBulks: get the total number of bulks	bulkThreadNum: get the total number of threads
getMyBulkId: get bulk id	bulkThreadId: get thread id
bspQsize: return the number of message still in queue	bulkThreadStart: start all threads
bspRemove: return and move an item of the queue	bulkThreadEnd: assure that all threads end
bspSync: barrier synchronization	bulkThreadSync: threads barrier synchronization

When users write a job using BSPCloud hybrid model, they should write two classes, one of which should extends class BspCloud and the other should extends class BulkTread. Users can define the operation through override the function bspOperate(). In the class bspOperate(), the compute complete by a set of BulkThreads. The hybrid programming approach is illustrated in Algorithm 1 and 2. In the Algorithm 1, we implement the matrix multiply through the operation bspOperate(). The compute phase of the bspOperate() is completed by a set of bulk threads. These bulk threads are start by the operation bulkThreadStart(). bulkThreadEnd() is used to assure that the compute tasks of bulk threads have been completed. The operation of the bulk thread is implemented by the method parallelComp() in the algorithm 2. When compare the Algorithm 1 with the Algorithm 2, we find that the two algorithms are almost the same except some APIs, which shows the BSPCloud hybrid programming model don't increase workload.

Algorithm 1 matrix multiply using the BSPCloud library

Calculates $C = A \times B$ in parallel.Based on the local data of every VM

```
Input:local m×k submatrix a of A
      local k×n submatrix b of B
public void bspOperate(){
1:int nBulk = getBspBulks();
2:int myBulkId = getMyBulkId();
3:int superStep = nBulk;
4:for s=1 to superStep do
5: bulkThreadStart(number);
6: bulkThreadEnd()
7:if myBulkId!=nBulk then
8:bspMp.send(myBulkId+1,
  arr,BspDataType.BSP_FLOAT);
9:else
10:bspMp.send(1, arr,BspDataType.BSP_FLOAT)
11:bspSync();
12:b = (BspArray<Float>)bspRemove()
}
```

Algorithm 2. The compute of shared-memory model using multiple bulk threads

```
Calculates the matrix multiply of submatrix a and submatrxi b
Input: ta submatrix of a
       tb submatrix of b
public void parallelComp(){
1: int tid = this.getTid();
2: int t = this.getT();
3: load(ta,tb)
4: for s=1 to t
5: compute ta×tb
6: if int t = this.getT() then
7: comm.put(tid-1, tb);
8: else
9: comm.put(this.getT()-1, tb);
10: bulkThreadSync();
11: tb = comm.move(tid);
}
```

5.2 BSPCloud Communicate Library and its Implementation

According to the different communication mechanism, we give two set of communication library.

5.2.1 Distributed-memory Communication Library and its Implementation

In our BSPCloud, distributed-memory communication uses massage passing mechanism. All communication operations of message passing are encapsulated into a class BspMP. A BulkTracker can send messages through the method send (), and a BulkTracker can get messages through the method bspRemove(). We implement BSP message passing use java socket. Each BulkTracker has a communication thread which acts as a server socked, and it begin to listen when the BulkTracker start. When a BulkTracker need to communicate with another, it creates a socket and sends the data to another BulkTracker through java socket. When the BulkTracker receives the data, it put the data in the queue which is used to receive data. The data which has been in the queue can be got by the BulkTracker in the next super-step.

5.2.2 Share-memory Communication Library and its Implementation

In our BSPCloud, the communication between bulk threads of shared-memory model occurred in memory. We have implemented the shared-memory communication mechanism. When one bulk thread needs to send data to others, it put the data into the memory through the method put (). In the next super-step, the bulk thread can get the data through the method get (). The public memory area is implemented by a class Global. A HashMap is defined in the class Global. The value of the HashMap stores the data which is put by the method put (), the key of the HashMap denote which the data is sent to. When a bulk thread sends the data to others, the data and id of the destination thread are stored in the HashMap. The data can be attained by the destination thread through id in the next super-step.

5.3 BSPCloud Submit

When a BSPCloud program have been written, it can be submitted to cloud platform uses the class BspJob which provides by BSPCloud programming library. The BspJob gives some application programming interfaces (APIs). Users can set the program's name and resources which they want to apply through these APIs, and submit the BspJob to cloud platform through the operation submit ().

The submit APIs are implemented in java socket. There is receiving job thread which is a server socket running in the BspJobTracker. When the operation submit () is invoked, it creates a socket. The information and the program which the user defined will send to BspJobTracker. The job not executes immediately, and it is sent to schedule. When the job fetch form schedule, it will be run in the cloud platform.

6. Experiment Evaluations

We have implemented BSPCloud in java. This section presents experimental results evaluting the performance predictable, speedup and scalability and the performance of BSPCloud affected by multi-core.

6.1 Experimental Setup

A set of experiments have been conducted by a virtualization cluster using three physical systems in our laboratory and the details of parameters are as follows:

host1: 8-core equipped with two 4-core Inter(R) Xeon(R) cpu E5620 running at 2.40GHz,32G memory.

host2:4-core AMD phenom Ⅱ X4 CPU B97 running at 3.20G Hz, 10G memory

host3: 4-core Inter(R) i5 processor running at 3.10GHz, 4G memroy.

All systems are running the 64-bit version of Ubuntu 10.04.4. The hosted virtualization is kvm. To evaluate the performance of BSPCloud, matrix multiplication experiments have been performed using different datasets.

6.2 Performance Predictability of BSPCloud

BSPCloud has the advantage of performance predictability. If the process speed of virtual machines communication bandwidth between VMs and synchronize latency of BSPCloud can be known, we can know the time cost of the application before it run. To estimate the performance predictability, we develop a benchmarking program which is implemented in BSPCloud library. Table 2 gives some parameters results.

Table 2. Benching Parameters

Shared-memory parameters				Distributed-memory parameters			
p	r (Mflop/s)	g (Kflops)	l (ms)	p	r (Mflop/s)	g (Kflops)	l (ms)
1	72.5	0	0	1	69.2	0	0
2	71.3	0	3	2	68.4	639	14
4	68.1	0	6	4	68.1	632	36
8	67.1	0	8	8	67.9	625	68

To evaluate the time cost predictable of BSPCloud, we use the matrix multiply in Algorithm 1. Shared-memory(SM) model and distributed-memory (DM) model are used respectively and the size of matrix A and B are both 4000×4000 in our experiment. For evaluate the shared-memory model, we apply three VMs, the number of virtual CPU (VCPU) are 2, 4 and 8 respectively. We use 2 threads, 4 threads and 8 threads respectively corresponding. For distributed-memory model, we apply 8 VMs and each has one core. We use 2 VMs, 4 VMs, and 8 VMs respectively. We compare the computing results with the theoretical values computing in equation (1). Figure 2 shows the results.

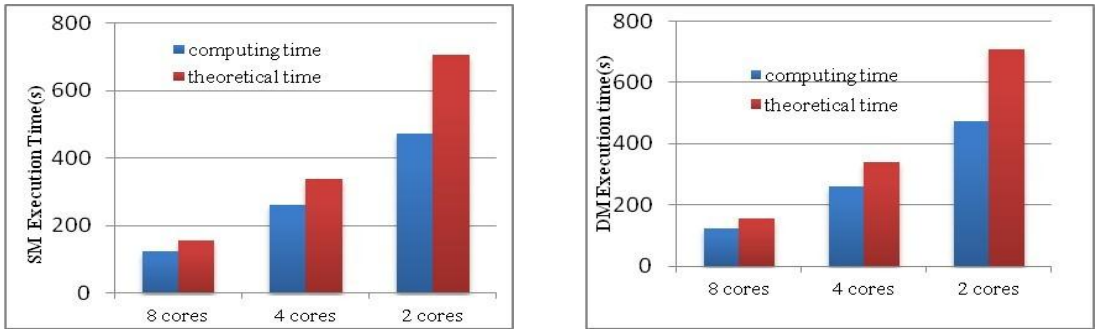
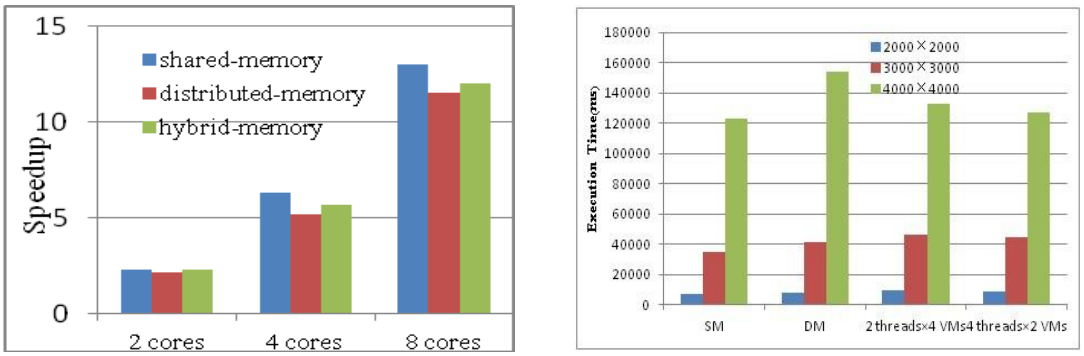


Figure 2. Time Cost Predictable Evaluate using Shared-memory and Distributed-memory

BSPCloud does not make very accurate prediction on the running time. This is because some factors have been overlooked. Firstly, the parameters in Table 2 are the average value, theses values are different in each run. Secondly, network bandwidth is measured by small data transfer, connection creates and disconnect time are not considered in the benchmarking. This lead to network bandwidth parameter is larger than real value. Thirdly, for the distributed-memory model, BulkTrackers begin compute when they receive the BspJobTracker's notification. However, BulkTrackers are hardly receiving notification in the same time, which will lead to boot latency. For the shared-memory model, computing time among threads is different which caused by operate system schedule. This will lead to wait latency in the synchronize phase.

6.3 Speedup and Scalability

We also evaluate the speedup and scalability of BSPCloud. We first fix the size of matrix A and B, they are both 4000×4000. For shared-memory model, we apply 3 VMs and the numbers of VCPU are 2, 4 and 8 respectively. We compute the matrix multiply use the 3 VMs respectively. For distributed-memory model, the matrix multiply is compute use 2 VMs, 4 VMs and 8 VMs respectively, each of these VMs apply one VCPU only. For hybrid distributed-memory and shared-memory model, we use one VM, 2 VMs and 4 VMs and each has two VCPUs. We create one thread in each VCPU in all our experiments. Figure 3(a) shows the speedup of three models. The x-axis is the total number of VCPUs (cores), the y-axis is the speedup.



(a) Speedup of BSPCloud. (b) Vary the size with fix VCPUs

Figure 3. Speedup and Scalability of BSPCloud

The speedup of shared-memory model is biggest in the three models. This is because the communication of shared-memory model only occurs in memory. The speedup of BSPCloud hybrid model is between distributed-memory and shared-memory. This is because the BSPCloud hybrid programming model uses hierarchical communication mechanism. Some data communication occurred in memory and some data communication through message passing mechanism. We find a surprising phenomenon form the Figure 3(a). The speedup of three models is all more than the total numbers of core. This is caused by the VM schedule. When a VM which locates in the multi-core host machine only has one VCPU, it is not always allocated physical CPU (PCPU).

We also fix the size of machines and change the size of the matrix. For shared-memory model, we create a VM with 8 VCPUs in an 8-core physical machine. For distributed-memory model, we create 8 VM which has one VCPU in the same 8-core physical machine. For hybrid model, we give two configurations. One is 2 VCPUs and

4 VMS, and the other is 4 VCPUs and 2 VMs. Figure 3(b) shows the results. The x-axis is the size of machines, and the y-axis is the execution time of the program. The scalability of the two configuration of the hybrid model is both between than distributed-memory and shared-memory. This is also because the hierarchical communication mechanism which is analyzed in the above. We also notice that the scalability of the configuration of 4 VCPUs and 2 VMs is better than the configuration of 2 VCPUs and 4 VMs. This is because the communication performance between VCPUs is better than VMs. Times New Roman is specified, Times Roman, or Times may be used. If neither is available on your word processor, please use the font closest in appearance to Times New Roman that you have access to. Please avoid using bit-mapped fonts if possible. True-Type 1 fonts are preferred.

7. Conclusions

In this paper, we propose a programming model for cloud computing environment—BSPCloud. BSPCloud is a hybrid of distributed-memory model and shared-memory model system. BSPCloud can make full use of multi-core clusters and has the advantage of performance predictability. The experiments results show that the cost time of BSPCloud can be predictable. We also show the speedup and scalability of BSPCloud in different configuration. We notice that the speedup and scalability are greatly influenced by virtual machine schedule in the cloud computing environment. In the next, we will integrate VM schedule into BSPCloud.

Acknowledgements

This work is supported by Innovation Action Plan supported by Science and Technology Commission of Shanghai Municipality (No.11511500200).

References

[1] J. L. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters”, Communications of the ACM, vol. 51, **(2008)**, pp. 107-13.
[2] M. Iiard, M. Budiu and Y. Yuan, “Dryad: distributed data-parallel programs from sequential building blocks”, Oper Syst Rev., vol. 41, **(2007)**, pp. 59-72.
[3] C. Olston, B. Reed and U. Srivastava, “Pig latin: a not-so-foreign language for data processing”, ACM, **(2008)**.
[4] A. Abouzeid, K. Bajda-Pawlikowski and D. Abadi, “HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads”, Proceedings of the VLDB Endowment, vol. 2, **(2009)**, pp. 922-33.
[5] Y. Yu, M. Isard and D. Fetterly, “DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language”, USENIX Association, **(2008)**.
[6] L. G. Valiant, “A bridging model for parallel computation”, Communications of the ACM, vol. 33, (1990), pp. 103-111.
[7] D. Righir, L. Pilla and A. Carissimi, “MigBSP: A Novel Migration Model for Bulk-Synchronous Parallel Processes Rescheduling”, New York: Ieee, **(2009)**.
[8] O. Bonorden, “Load balancing in the bulk-synchronous-parallel setting using process migrations”, 2007 IEEE International Parallel and Distributed Processing Symposium, **(2007)**.
[9] M. A. H. Hassan and M. Bamha, “parallel processing of ‘group-by join’ queries on shared nothing machines”, Springer-Verlag New York Inc., **(2008)**.
[10] V. G. Costa, A. Prinrista and M. Marin, “A parallel search engine with BSP”, IEEE, **(2005)**.
[11] Hadoop, http://hadoop.apache.org/.
[12] M. Gregorz, M. H. Austern, J. C. B. Aart, H. llan, L. Naty and C. Grzegorz, “Pregel: A system for large-scale graph processing”, Proceedings of the 2010 International Conference on Management of Data, **(2010)**, pp. 135-145.
[13] S. Seo, E. J. Yoon, J. Kim, S. Jin, J. -S. Kim and S. Maeng, “HAMA: An efficient matrix computation with the MapReduce framework”, The 2nd IEEE International Conference on Cloud Computing Technology and

Science, **(2010)**, pp. 721-726.
[14] L. G. Valiant, "A bridging model for multi-core computing", Journal of Computer and System Sciences, vol. 77, **(2011)**, pp. 154-166.
[15] A. N. Yzelman and R. H. Bisseling, "An object-oriented bulk synchronous parallel library for multi-core programming", Concurrency and Computation-Practice & Experience, vol. 24, **(2012)**, pp. 533-553.

Authors

Xiaodong Liu

He is currently a Ph.D. student of Shanghai University. His primary research interests cover virtualization, cloud computing and grid computing.

Weiqin Tong

He received his Ph.D. degree from Shanghai Jiao Tong University. He is currently a Professor of Shanghai University. His primary research interests cover High performance computing, embedded system and distributed system.

Fu Zhi**ren**

He received his Ph.D. degree from University of Science and Technology of China. He is currently a senior engineer of China Telecom Corporation Ltd. His primary research interests cover computer network and cloud computing.

Liao Wenzhao

He is currently an engineer of China Telecom Corporation Ltd. His primary research interests cover computer network and cloud computing.