Vol.2010-HPC-126 No.40
2010/8/5

Evaluation of Natural Language Processing on Local and Public Cloud Resources

Hao Sun †1 and Kento Aida†2,†1

In high-performance computing, parameter survey applications, which consists of a lot of independent tasks with different input parameters are typical and conventionally run on local computing resources with batch scheduler systems such as SGE and Condor. Our targeted PSA application is a Natural Language Processing (NLP) application. It uses Support Vector Machine (SVM) to learn and eliminate abuse words from the commercial bulletin board systems (BBSs) documents. Security issues become important because it processes confidential documents including personal informations. In this paper, we implemented an effective and secure hybrid mechanism with our previous work, InterS. It adds IaaS Cloud resources to the batch schedulers when the local resources are insufficient due to failures or overhead by external application tasks. It utilizes Amazon VPC service to enable the secure computation of our NLP application. InterS run the NLP application in our experiment and the evaluation of time efficiency is shown from the results.

1. Introduction

A parameter survey application (PSA) is known as a typical application running on high-performance computing systems. PSA consists of a lot of independent tasks with different input parameters, and tasks are executed in parallel on different CPU (cores). A batch scheduler, e.g. SGE[1)] and Condor[2)] is used to run tasks on local computing resources such as a PC cluster operated in the user site. In this case, tasks are submitted to the local batch queue and the batch scheduler dispatches tasks to free CPU (cores) in the local resource pool. Natural Language Processing applications usually run machine learning tasks in the PSA style. Our target NLP application is developed to detect and eliminate abuse words in documents on the internet bulletin board systems (BBSs). It runs the machine learning method, the Support Vector Machine (SVM), to learn input sample documents and then produces the high quality model to detect abuse words. The machine learning method needs to evaluate huge number of different SVM parameters. The input documents include personal information, thus, the computation needs to be executed in secure environment.

Infrastructure-as-a-Service Cloud (IaaS Cloud) providers such as Amazon[3)] provides computational resources to users in an on-demand manner by virtualizing their physical resources. A user purchases the (virtualized) resources in a pay-as-you-go fashion and usually is charged in an hourly basis. Security is one of issues[4)] in the current IaaS Cloud environment. In some cases, input files of an application may include confidential data that should be protected from other users, e.g. anonymous data or data including personal information.

Some batch schedulers, e.g. SGE and Condor, currently support job submission interfaces for Amazon EC2. Also, the hybrid execution mechanism and the scheduling performances have been studied in[5),6)]. However, the discussions are limited to the hybrid execution mechanism. To the best of our knowledge, no previous work discusses a mechanism and performance to run PSA in the hybrid and secure manner.

In this paper, we propose a hybrid and secure execution mechanism to run the NLP application with both local computing resources and IaaS Cloud resources. The proposed mechanism enables a user to submit NLP tasks through the local batch scheduler, or SGE. The submitted tasks run on local computing resources when sufficient resources are available in the local resource pool. Also, the proposed mechanism controls resources so that the submitted the application meet the deadline. When local computing resources are not available by failures or overload due to external application tasks, the proposed mechanism automatically creates virtual machine (VM) instances in IaaS Cloud (Amazon EC2) and add the VM instances into the resource pool of the local batch scheduler. Also, the proposed mechanism deletes the VM instances in the resource pool when sufficient local resources are available. The Amazon Virtual Private Cloud (VPC)[7)] service is utilized to enable secure computation of the target application. The communication between the local computing resources and Amazon EC2 is securely performed by establishing virtual private network between both sites.

†1 Tokyo Institution of Technology
†2 National Institute of Informatics

We implemented the proposed mechanism in our job scheduler, InterS[8], and conducted experiments by running the natural language processing application (NLP). Our experimental results show that the proposed mechanism enable hybrid and secure execution of NLP utilizing local computing resources and IaaS Cloud.

The rest of the paper is organized as follows: Section 2 discusses related work, and our target application, NLP, is introduced in Section 3. The proposed mechanism and its implementation are presented in Section 4. Section 5 shows the experimental results, and Section 6 summarizes our work and outlines the future work.

2. Related Work

This section reviews related work. First we review batch schedulers that support interfaces to IaaS Cloud, or Amazon EC2. Then performance studies of scheduling algorithms for hybrid resources, local and public cloud resources are presented.

2.1 Batch Schedulers with Amazon EC2 Interface

Sun Grid Engine (SGE) is distributed resource management software. It is widely used as a batch scheduler of local resources, e.g. PC clusters. SGE 6.2 enables a user to submit jobs to resources in Amazon EC2 through the Hedeby Grid Engine service adapter[9),10)].

Condor is a workload management system, which dispatches user tasks to resources using the matchmaking mechanism[11)]. It is used as a batch scheduler on not only local resources, e.g. a computing resource pool in a local site, but also distributed computing platforms such as the grid. Condor also supports the Amazon EC2 interface and it enables hybrid execution of tasks between local resources and Amazon EC2 resources by creating instances from the pre-configured Amazon EC2 AMI for the Condor worker process.

Both SGE (with Hedeby) and Condor work as batch schedulers. Although they improve overall system performance, e.g. throughput, performance of each application is sometimes sacrificed. Also, a user needs to create VM instances and save them in Amazon S3/EBS, that is, the user need to pay for the storage usage of Amazon S3 even when he/she does not run tasks.

The goal of the proposed mechanism is to enable application scheduling with hybrid and secure execution of PSA(s). It run PSA tasks to guarantee QoS, e.g. deadline, by utilizing both local resources and Amazon EC2 resources in hybrid and secure manner. Also, in the proposed mechanism, VM instances for Amazon EC2 are created and uploaded at runtime, and the instances are deleted when the application finishes. Thus, it minimizes the cost for using Amazon services.

2.2 Scheduling Algorithms for Hybrid Resources

Scheduling algorithms for hybrid resources have been studied in literatures[5),6)]. The work in[5)] proposes two kinds of market-oriented scheduling policies for hybrid execution and evaluated the efficiency with the Gridbus broker. In[6)], the authors discuss the cost benefit of six scheduling strategies to utilize IaaS Cloud resources with local batch schedulers and shows the improvement of local system response times. While the focus of the above work is to discuss performance of scheduling algorithms on hybrid resources, the focus of this paper is to propose a hybrid and secure task execution mechanism. Both work are complementary, i.e. the scheduling algorithms discussed in[5),6)] could be used on the proposed mechanism in this paper.

3. Target Application and Requirements

This section presents an overview of the application program used as the benchmark in this paper.

A bulletin board system (BBS) and a social network system (SNS), such as Facebook[12)] and Mixi[13)], are widely used in the world. One of problems in these systems is abuse words posted in the systems. They may include information that is contrary to public policy or personal information. Administrators of these systems try to eliminate the abuse words in their systems; however, it is too hard to eliminate the words effectively. We used the natural language processing application, or NLP, as the benchmark in this paper. NLP is designed to help people to eliminate abuse words efficiently and quickly. NLP runs in two phases, the learning phase and the elimination phase. Figure 1 shows that in the the learning phase, the NLP reads sample input documents collected from BBS as training data and learns using the machine learning method, or the Support Vector Machine (SVM). The annotation data, which identifies abuse words, are

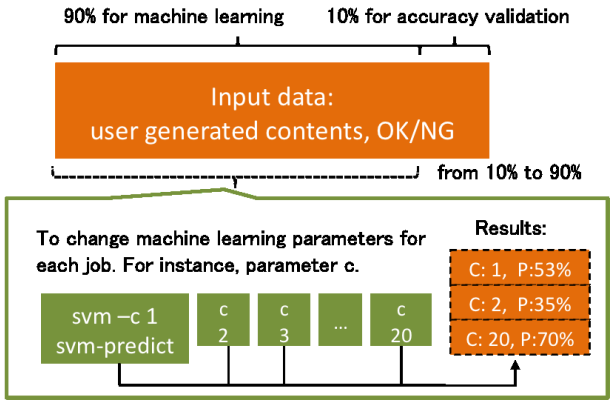

Fig. 1 Overview of the NLP application

also given to NLP with the training data. Figure 1 also shows that *90%* of the input documents are used for training and *10%* of the data are used for validation of the learning results. Finally, the model to eliminate abuse words is generated. In the elimination phase, the generated model is used to eliminate abuse words in the target documents.

In the experiments, we collected training data from six or seven commercial BBS/SNS in Japan and ran the learning phase of NLP with the collected training data. SVM is implemented as PSA, that is, it runs training tasks using different SVM parameters. We used the SVM parameter ranging from *1* through *100*. Running one training task spend about *50* minutes on a PC with *2.4*GHz (one core of a dual core AMD Opteron^{tm} processor)

The training data, or documents in commercial BBS/SNS, may include anonymous or personal information, e.g. addresses and phone numbers; thus, we need to run NLP in secure environment. Also, we need to run NLP with a batch scheduler, e.g. SGE, because NLP is implemented as scripts to submit array jobs thorough SGE.

4. Proposed Method

This section outlines the proposed mechanism and presents implementation in details. The proposed mechanism efficiently utilizes both local computing resources and IaaS cloud resources, or Amazon EC2, to meet a deadline of the summated PSA. The proposed scheduling mechanism is implemented in our job scheduler, or InterS.

A user submits a PSA, or tasks in the PSA, to InterS indicating the deadline to finish the PSA. Then, InterS runs a part of tasks in the submitted PSA fully utilizing local computing nodes. After the tasks finish on the local computing nodes, InterS investigates execution time of the finished tasks and estimates the time to complete all tasks in the submitted PSA. If the estimated completion time is before the deadline, InterS runs the rest of tasks on local computing nodes. If the estimated completion time is after the deadline, InterS computes the number of external computing nodes, or VM instances in Amazon EC2, required to complete all tasks before the deadline and runs the rest of tasks both on local computing nodes and VM instances in Amazon EC2. The communication between the local computing nodes and the VM instances in Amazon EC2 is securely performed by the Amazon VPC service.

The rest of this section presents detailed implementation of the proposed mechanism: (1) the hybrid resource management, (2) resource extension and reduction and (3) secure communication with Amazon VPC.

4.1 Hybrid Resource Management

InterS works as a frontend interface of SGE. InterS configures two queues in SGE, *all.q* and *ec2.q*, to submit tasks to local resources and Amazon EC2, respectively. When resources in Amazon EC2 resources are required, InterS initiates VM instances in the VPC subnet and add VM instances into *ec2.q*. Communication between InterS and SGE is implemented using the Distributed Resource Management Application API (DRMAA)[14]. Figure 2 shows an example of scripts running in InterS to add Amazon EC2 resources to *ec2.q*.

- Preparation
  InterS uses Amazon EC2 command line tools to upload pre-configured AMI files to Amazon S3 and then register the VM image (line 1 in Figure 2). This step requires user credentials such as Amazon service access keys: a_key, s_key and upload destinations, e.g. $bucket$, $mani\_file$.
- Provisioning

```
// AMI preparation
1. ec2-upload-bundle -b $bucket -m $mani_file -a $a_key -s $s_key
2. ec2-register $bucket/$mani -n $bucket

// Extension
3. ./inst_sge -x -auto $configure_template_file
4. qconf -aattr hostgroup hostlist $hostname @ec2hosts
5. qconf -aattr queue slots [$hostname=$numOfCores] ec2.q
6. ssh root@$hostname sudo -usgeadmin /etc/init.d/sgeexecd.inters-ec2 start
```

Fig. 2 Methods to extend SGE resources

```
// Uninstall
1. qconf -dattr hostgroup hostlist $hostname @ec2hosts
2. qconf -dattr queue slots [$hostname=$numOfCores] ec2.q

// AMI deletion
1. ec2-deregister $ami_id
2. ec2-delete-bundle -a $a_key -s $s_key -b $bucket -p $mani_file
```

Fig. 3 Methods to reduce SGE resources

InterS uses Amazon APIs to initiate Amazon EC2 resources.

- Extension
  InterS creates template files and installs the SGE system with the automatic installation method[15]. To generate the installation template file, configurations of hostname, SGE_ROOT, SGE_CELL and other values should be collected for each Amazon EC2 resource. After the initiation of Amazon EC2 resources, InterS installs SGE systems with the configuration files by command *inst_sge* (line *3* in Figure 2). To utilize the new Amazon EC2 resource, InterS should add it to the proper host group and queue of SGE, *@ec2hosts* and *ec2.q* are used in our implementation. Finally, InterS starts SGE service through ssh.

When Amazon EC2 resources are no longer required, InterS deletes the VM instances from *ec2.q*, shutdowns the VM instances and deletes the AMI files from Amazon S3 resources. Figure 3 shows an example of scripts to delete Amazon EC2 resources.

4.2 Resource Extension and Reduction

InterS periodically checks statuses of queued tasks and adds/deletes resources

Table 1 Summaries of Amazon EC2 Instance Types

Type	Cores	Capacity [GHz]*1	RAM [GB]	Cost [$/h]
m1.large	2	2 (4)	7.5	0.34
m1.xlarge	4	2 (8)	15.0	0.68
m2.xlarge	2	3.25 (6.5)	1.7	0.50
m2.2xlarge	4	3.25 (13)	7.5	1.20
m2.4xlarge	8	3.25 (26)	15.0	2.40
c1.xlarge	8	2.5 (20)	7.0	0.68

*1. (n) stands for the number of ECUs, the CPU capacity unit for Amazon EC2 service.

```
   C_ec2 = MAXIMUM
   N_t_local = ⌊N_localslots T_remain / T_local⌋
   if N_t_now > N_t_local then
     for cpu_capacity in all Capacity [GHz] of Table 1 do
5:     if T_cpu_capacity = 0 then
         T_cpu_capacity = T_local × CPU_local / cpu_capacity
       end if
       N_ec2slots = ⌈(N_t_now − N_t_local) T_cpu_capacity / T_remain⌉
       S_cpu_capacity = planEC2Resources(N_ec2slots, cpu_capacity)
10:    C_ec2now = calculateCost(S_cpu_capacity)
       if C_ec2now < C_ec2 and C_ec2now ≤ Budget then
         S_ec2 = S_cpu_capacity
       end if
     end for
15: end if
   // To extend resources
   initiateEC2Resource(S_ec2)
   moveQueuingTasksToEC2(N_t_now − N_t_local)
```

Fig. 4 Resource Extension Algorithm

to execute the tasks so that all tasks finish before the deadline. In the current implementation, we configure InterS to do the above procedures every *60* seconds. Amazon EC2 offers multiple types of VM instances as shown in Table 1. InterS chooses the cheapest one that completes all tasks before the deadline.

Figure 4 shows the algorithm for resource extension, or adding VM instances in Amazon EC2 into *ec2.q*. First, InterS obtains the number of queued tasks N_{t_now}. The queued tasks means tasks both running in resources and waiting in queues. Then, InterS estimates N_{t_local}, the number of queued tasks that can finish before the deadline in the local resources. Line 3 shows the equation to compute N_{t_local}. $N_{localslots}$ indicates the number of available cores in the local resources. T_{remain} means time to the deadline computed by $Deadline - CurrentTime$ and T_{local}

indicates the average execution time of tasks that ran on the local resources. If N_{t_now} is greater than N_{t_local}, InterS adds Amazon EC2 resources and estimates costs for the Amazon EC2 resources. InterS chooses one or more instance types with the same Capacity [GHz] (in the third column in Table 1) for the Amazon EC2 resources. There might be multiple options to choose instance type(s) for the Amazon EC2 resources. InterS compute a total cost for VM instances for each option and chooses the cheapest option under the user budget. (This algorithm is presented from the line *5* through *15*). We implemented functions presented in Table 2 on InterS. (They are indicated in the boldface in Figure 4).

The number of VM instances, which are added to *ec2.q*, should be appropriately decided so that all tasks finish before the deadline. The line *10* in Fig. 4 computes the number of CPU cores (in VM instances) added to *ec2.q*, $N_{ec2slots}$. $T_{cpu\_capacity}$ indicates the average task execution time on VM instances of the "cpu_capacity" type in the Amazon EC2 resources. $N_{ec2slots}$ is computed for all VM instance types in Table 1 and the cheapest type is chosen (line *13*).

Figure 5 presents the algorithm to delete VM instances from *ec2.q*. When a task running on an Amazon EC2 resource finishes, InterS checks if other tasks are associated to the same VM instance as the finished task (Line *4* through line *7*). If no tasks are associated to the VM instance, or the VM instance is no longer needed, InterS sets the shutdown flag in the VM instance from the line *8* through *9*. InterS does not shutdown the flagged instance at this time. A VM instance in Amazon EC2 is charged for a slot of *60* minutes duration. Thus, InterS delays the shutdown by the end of the charged slot, so that it avoids frequent shutdown and initialization of VM instances and paying extra cost.

4.3 Secure Communication with Amazon EC2

We implemented InterS with Amazon VPC to establish secure communication between local resources and Amazon EC2 resources. Amazon VPC is a secure and seamless bridge between local infrastructure and the AWS Cloud[7]. Table IV summarizes the prices for the VPC service. In the implementation, local resources and Amazon EC2 resources communicate through gateways, the customer gateway associated with the local resources and the VPC gateway associated with Amazon EC2 resources. The secure communication between the gateways is established via the IPSec-VPN tunnel[16]. To establish IPSec-VPN tunnels between local cluster and Amazon VPC, the open source software ipsec-tools[17] and quagga[18] are used[19]. Parts of the configuration are shown in Figure 6. The *racoon.conf* configures IPSec connection algorithms and other connection informations (e.g Line *1* shows the IP address of the VPC gateway), the

Table 2 Functions for Resource Extension

Function	Details
planEC2Resources($N_{ec2slots}$, $cpu\_capacity$)	Return Amazon EC2 resources such as [(m1.xlarge,1)] when $N_{ec2slots}$ is 3 and *cpu_capacity* is 2GHz
calculateCost($N_{cpu\_capacity}$)	Return the cost of the Amazon EC2 resources. e.g returns $0.68 for [(m1.xlarge,1)] for one hour.
initateEC2Resource(S_{ec2})	Create Amazon EC2 resources and extend SGE resources. S_{ec2} is the Amazon EC2 resources set, such as [(m1.xlarge,1)]
moveQueuingTasksToEC2($N_{ec2tasks}$)	Move tasks from *all.q* to *ec2.q*.

Table 3 Functions for Resource Extension

Function	Details
waitAnySGETasks()	Return a finished task
getEC2Resource(J)	Return the corresponding Amazon EC2 resources of task J.
getUnfinishedJobsWithResource(R)	Return all unfinished tasks with the Amazon EC2 resource R.
markEC2ResourceShutdown(R)	To mark resource R status as shutdown.
deleteTask(J,R)	To delete task J from the corresponding resource R.

```
   loop
     // Detect if any job finished
     J = waitAnySGETasks()
     if J is a task running on Amazon EC2 resource then
5:     // Get the EC2 resource handle of job J
       R_j = getEC2Resource(J)
       J_r = getUnfinishedJobsWithResource(R_j)
       if sizeof(J_r) == 0 then
         markEC2ResourceShutdown(R_j)
10:    else
         deleteTask(J, R_j)
       end if
     end if
   end loop
```

Fig. 5 Resource Reduction Algorithm

Table 4 Summaries of Amazon VPC Pricing

VPN Connection [h]
$0.05

VPN Data Transfer In [$/GB]
free until 11/01/2010 $0.10 after 11/1/2010

VPN Data Transfer Out [$/GB]	
First 1 GB/Month	$0.00
Up to 10 TB/Month	$0.15
Next 40 TB/Month	$0.11
Next 100 TB/Month	$0.09
Over 150 TB/Month	$0.08

ipsec-tools.conf configures the rules when to use IPSec-VPN for communication. For instance, through line *1* to line *4*, it shows the rule: when send packets from the customer gateway to the Amazon VPC gateway, secure it with VPN and the secure algorithms and parameters are used from the lines *14* to *17* in *racoon.conf*.

InterS uses the Java API of Amazon EC2 to manipulate Amazon VPC service. There are mainly six steps to initialize and finalize the VPN connection: to create VPC and subnets, then to create the customer gateway, VPN gateway and VPN connections between them. Finally, InterS attach the VPN gateway to the Amazon VPC. InterS detects the required information and do the above works automatically. Users just need to make sure the required packages: *ipsec-tools* and *quagga* are installed properly. Amazon VPC uses Pre-Shared-Key method for the authentication and to setup such credentials requires root privilege of the customer gateway, InterS uses *sudo* command to avoid this kind of problems. Communication between resources in Amazon VPC and resources outside of Amazon VPC needs to be performed through the customer gateway[7]. This may cause serious performance degradation when resources in Amazon EC2 share files in Amazon S3. Let us suppose that Amazon EC2 and S3 resources are operated in a datacenter in US and local resources are located in the other country, e.g. Japan. In the case that two VM instances in Amazon EC2 access files in Amazon S3, data for the file access is transferred through the customer gateway in Japan. We do not use Amazon S3 to avoid the performance degradation.

```
# /etc/racoon/racoon.conf
 1. remote 72.21.209.225 {
 2.         exchange_mode main;
 3.         lifetime time 28800 seconds;
 4.         proposal {
 5.                 encryption_algorithm aes128;
 6.                 hash_algorithm sha1;
 7.                 authentication_method pre_shared_key;
 8.                 dh_group 2;
 9.         }
10.         generate_policy off;
11. }
12. sainfo address 169.254.255.2/30 any address
13.                         169.254.255.1/30 any {
14.     pfs_group 2;
15.     lifetime time 3600 seconds;
16.     encryption_algorithm aes128;
17.     authentication_algorithm hmac_sha1;
18.     compression_algorithm deflate;
19. }

# /etc/ipsec-tools.conf
1. spdadd 169.254.255.2/30 169.254.255.1/30 any -P out ipsec
    esp/tunnel/136.187.33.82-72.21.209.225/require;

2. spdadd 169.254.255.1/30 169.254.255.2/30 any -P in ipsec
    esp/tunnel/72.21.209.225-136.187.33.82/require;

3. spdadd 192.168.200.0/24 169.254.255.2/30 any -P in ipsec
    esp/tunnel/72.21.209.225-136.187.33.82/require;

4. spdadd 169.254.255.6/30 192.168.200.0/24 any -P out ipsec
    esp/tunnel/136.187.33.82-72.21.209.193/require;
```

Fig. 6 Configurations of IPSec-VPN connection

We put a NFS server in Amazon VPC for file sharing among VM instances in Amazon EC2.

5. Experimental Studies

Table 5 Experimental Settings

(a) Local Resource Information

8 SGE execution nodes, 4 cores (2.4GHz AMD) in each node.

(b) Summarization of Task Workloads

# NLP Tasks	AET*1 [s] on 2.4GHz	AET [s] on 2GHz	AET [s] on 3.25GHz
100	2564.90 (±266.05)	2689.50 (±233.79)	2923.6 (±1043.21)

*1 AET stands for average execution times.

In this section, the proposed method is evaluated and the execution time and the costs of utilizing Amazon EC2 are discussed. The NLP application user executes *100* tasks in a local cluster with *32* cores (2.4GHz AMD) within *9000* seconds. To guarantee the deadline, InterS used three kinds of resources with

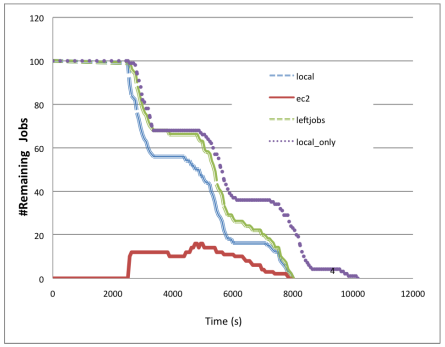

Fig. 7 Job Distribution with Hybrid Resources

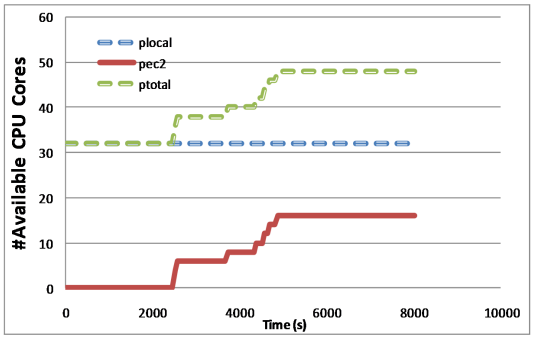

Fig. 8 Amazon EC2 Resource Allocation

different CPU capacities: the local cluster resources and *m1.**, *m2.** resources in Amazon EC2. We assume one EUC as one 1GHz CPU, so *m1.** types of Amazon EC2 resources have *2*GHz CPUs and *m2.** types have *3.25*GHz. The details of the tasks and local cluster information are summarized in Table 5. The execution time with each CPU capacity is calculated as the average of *100* different tasks. From the results, we can see the execution time do not change linearly between different CPU types, so the execution time estimation is also a challenge. Since evaluation of the hybrid execution method efficiency is the aim of this experiment, we left the problem to our future works.

We will show the scheduling performance of our proposed method below.

Figure 7 shows the task distribution of local resources and Amazon EC2 resources. The x axis stands for elapsed time, and the y axis stands for the number of remaining tasks. There are four lines in each figure, and the meaning of them are as follows: line *local* shows *locally running tasks* and line *ec2* shows that of Amazon EC2 resources, line *leftjobs* stands for *remaining tasks* and is the sum of *local* and *ec2*. To compare the scheduling performance with local only method, the *local_only* line is added to show the number of remaining tasks. We can see that InterS adds Amazon EC2 resources and makes the total execution time shorter than the *local_only* one.

InterS calculates the average of finished tasks, and regards it as the execution times of further tasks. InterS needs the execution information to extend local resources, so it can not start hybrid execution until any one of the tasks finish. The first *2000* seconds in the front of Figure 7 and Figure 8 shows the waiting time of the first finished task and the time of initializing the first Amazon EC2 instance. Table 6 shows the resource extension steps for the NLP tasks. InterS uses the resource extension algorithm and detects that only *91* tasks can be finished within the remaining *6631* seconds, but *99* tasks are running with *32* cores in local resources. Thus, *8* tasks should be executed with Amazon EC2 resources and until the deadline, the *m1.xlarge* type of Amazon EC2 instance with four cores can executes two cycle before the deadline, so InterS allocates two of them and assigns the *8* tasks to the resources. InterS repeated such process to guarantee the QoS of the applicaion. And finally, the *100* tasks cost *8065* seconds to finish and takes *$5.44* to allocate eight Amazon EC2 instances. We can see that InterS successfully guaranteed the deadline.

Table 6 Resource Extension Details

Time Remain [s]	#RJ*1/cores (local or ec2)	#JCF*2	(#Tasks, Resources)
6631	99/32 (local)	91	(8,m1.xlarge*2)
6565	89/32 (local)	88	(4,m1.large*2)
4753	56/32 (local)	55	(2,m1.large)
4571	54/32 (local)	53	(2,m1.large)
4450	52/32 (local)	51	(2,m1.large)
4268	50/32 (local)	49	(2,m1.large)

*1. RJ stands for remaining jobs.
*2. JCF stands for jobs can finish within the remaining time.

In Figure 8, the x axis stands for elapsed time and the y axis indicates the number of available cores. *Plocal* shows the available cores in local clusters and *pec2* shows that of Amazon EC2 resources. The line *pec2* in Figure 8 shows that InterS allocated Amazon EC2 resources six times (Table 6) and It only allocated resources when required. Finally, InterS utilized eight Amazon EC2 instances and guaranteed the deadline successfully.

6. Conclusions

In this paper, we proposed a hybrid and secure execution mechanism of PSA applications on local and public Could resources. It can help users to guarantee their deadline as while as providing a secure execution environment. We implemented the system by adding three functions to our job scheduler, InterS: (1) the hybrid resource management, (2) resource extension and reduction and (3) secure communication with Amazon VPC. Out target PSA is a natural language processing application, it uses the Support Vector Machine method to learn documents from commercial BBS/SNS systems, and then creates high quality models for detecting abuse words on the internet BBS/SNS(s). In our experimental study, a user runs much more tasks than the number of available cores in the local resources. From the results, we can show that the proposed method can allocate resources from Amazon EC2 properly and successfully guarantee the deadline. For the future works, we need to improve the accuracy of execution time estimation so as to provide a more cost effective mechanism.

Acknowledgment

A part of this work is supported by Japan Society for the Promotion of Science (JSPS) within the framework of Global COE Program "Photonics Integration-Core Electronics".

References

1) Gentzsch, W.: Sun Grid Engine: Towards Creating a Compute Power Grid, *In CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, IEEE Computer Society, p.35 (2001).
2) Online: Condor Project Homepage. `http://www.cs.wisc.edu/condor`.
3) Online: Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`.
4) Rittinghouse, J.W. and Ransome, J.F.: *Security in the Cloud*, chapterSecurity in the Cloud, pp.153–154, CRC Press (2009).
5) Salehi, M.A. and Buyya, R.: *Adapting Market-Oriented Scheduling Policies for Cloud Computing*, chapter Syntax-directed program modularization, pp. 351–362, Springer Berlin / Heidelberg (2010).
6) de Assuncao, M.D., di Costanzo, A. and Buyya, R.: Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters, *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, New York, NY, USA, ACM, pp.141–150 (2009).
7) Online: Amazon Virtual Private Cloud (Amazon VPC). `http://aws.amazon.com/vpc/`.
8) Sun, H. and Aida, K.: Interactive Application Scheduling with GridRPC, *IPSJ Transaction on Advanced Computing System*, Vol.3, pp.88–100 (2010).
9) Online: hedeby: Hedeby Project Overview. `http://hedeby.sunsource.net/`.
10) Online: SGE Hedeby and Amazon EC2. `http://wiki.gridengine.info/wiki/index.php/SGE-Hedeby-And-Amazon-EC2`.
11) Raman, R., Livny, M. and Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing, *In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pp. 28–31 (1998).
12) Online: Facebook. `http://www.facebook.com/`.
13) Online: ソーシャル・ネットワーキング サービス [mixi(ミクシィ)]. `http://mixi.jp`.
14) Online: Distributed Resource Management Application API Specification 1.0. `http://www.ggf.org/documents/GWD-R/GFD-R.022.pdf`.
15) Online: Sun Grid Engine: Automating the Installation Process. `http://wikis.sun.com/display/GridEngine/Automating+the+Installation+Process`.
16) Online: Guide to IPsec VPNs. `http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/nist80077.pdf`.
17) Online: IPSec-Tools. `http://ipsec-tools.sourceforge.net/`.
18) Online: Quagga Routing Suite. `http://www.quagga.net/`.
19) Online: Amazon VPC with Linux. `http://openfoo.org/blog/amazon_vpc_with_linux.html`.