A Hybrid and Secure Mechanism to Execute Parameter Survey Applications on Local and Public Cloud Resources
Parameter Survey Applications (PSAs) are typical HPC applications
- Natural Language Processing (NLP) application
- Batch schedulers such as SGE, Condor are used to allocate resources for PSAs
  - Resources are shared, QoS is hardly guaranteed
- Infrastructure-as-a-Service (IaaS) Cloud provides virtual resources for PSAs
  - Resources are dedicated to users
  - Security issues still exist
A NLP application to detect abusive words in SNS/BBS systems

- It uses SVM (Support Vector Machine) to learn documents collected from several services

Problems:

- Plenty of tasks
  - One task spends 45min with 2.4GHz processor.
  - There are hundreds of such tasks in one experiment (4 days).

- Secure execution environment
  - Input documents contain personal information.
To propose a hybrid and secure execution mechanism for the target PSA

A. Hybrid Resources Management
   - VM image preparation/deletion
   - VM instance initialization/finalization
   - Batch scheduler resource extension/reduction

B. Resource Utilization Algorithms
   - Algorithms of resource extension and reduction
   - How to guarantee QoS of PSAs with hybrid resources

C. Secure Communication
   - How to secure communication
     - Amazon VPC (IPSec-VPN), SSL-VPN, SSH
Batch Schedulers with Amazon EC2 Interface

- SGE+Hedeby: Service Level Objectives (SLO’s)
  - Entire system performance, not application
  - VM images are prepared by administrators
- Condor: Matchmaking
  - Not secure enough for our NLP application
  - VM images are prepared by administrators

Scheduling Algorithms for Hybrid Resources

- They are complementary work with our mechanism.
- Our research proposes a Hybrid and secure mechanism.
Overview of the hybrid Mechanism

NLP tasks with deadline

Proposed Mechanism

To guarantee QoS:
- total exec. time estimation
- move queuing tasks from all.q to ec2.q

Shared local resources. (all.q) network 192.168.0.0/24

Local resources (SGE nodes)

Customer GW

SGE tasks

Task I/O

VPC GW

Amazon EC2 resources

Private Cloud resources. (ec2.q) network 192.168.200.0/24

Amazon VPC

IPSec-VPN Connection

A

Resources extension

B

C

DRMAA
To manipulate hybrid resources, we provide three functions

- **VM image preparation/deletion**
  - Upload and register VM image to S3
  - Remove and un-register VM image from S3

- **VM instance initialization/finalization**
  - Create or Delete VM instances with Amazon EC2 APIs

- **Batch scheduler resource extension/reduction**
  - Add or remove VM instances from SGE
B. Resource Utilization Algorithms

- **Step 1: exec. time estimation (QoS guarantee)**
  - For each task: \( T_{local} \times CPU_{local} = T_{ec2} \times CPU_{ec2} \)
  - \( T_{local} \) is the average of all finished local tasks

- **Step 2: resource allocation**
  - Calculate the number of required CPU cores
  - Choose the cheapest group of VMs

- **Step 3: resource utilization**
  - Move local tasks to Cloud VMs

- **Step 4: resource reduction**
  - Finalize VMs before the end of instance-hour (60min)
C. Secure Communication

- Amazon VPC is used
  - There is no **standard API** of SSL-VPN software
  - NAT or **firewall problems** with SSH connections

- Implementation
  - Use Java API to manipulate Amazon VPC
  - Establish IPSec tunnels with IPSec-tools
  - Reconfigure IPSec tunnels with sudo command
    - PSKs (Pre-Shared-Key) are varying for each initialization
Overview of Experimental Study

- **Amazon VPC Overhead**
  - Six steps for Amazon VPC initialization
  - Five steps for Amazon VPC Finalization

- **Resource Extension Overhead**
  - VM image preparation / deletion
  - VM startup / shutdown
  - SGE installation / uninstallation

- **Performance of Resource Extension**
  - Scenario A: To guarantee QoS
  - Scenario B: To guarantee QoS while running with external jobs
Amazon VPC Overhead (1)

This overhead is acceptable, because the execution time of NLP tasks spends hours to days.

<table>
<thead>
<tr>
<th>Initialization Overhead [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VPC</strong></td>
</tr>
<tr>
<td><strong>Subnets</strong></td>
</tr>
<tr>
<td><strong>Customer Gateway</strong></td>
</tr>
<tr>
<td><strong>VPN Gateway</strong></td>
</tr>
<tr>
<td><strong>VPN Connections</strong></td>
</tr>
<tr>
<td><strong>Attachment</strong></td>
</tr>
<tr>
<td><strong>Total Cost</strong></td>
</tr>
</tbody>
</table>

The value is the average of 10 measurements.
Amazon VPC Overhead(2)

This overhead is acceptable, because the execution time of NLP tasks spends 50min.

<table>
<thead>
<tr>
<th>Finalization Overhead [s]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Subnets</td>
<td>2.63(±0.26)</td>
</tr>
<tr>
<td>Detachment</td>
<td>42.28(±23)</td>
</tr>
<tr>
<td>VPC</td>
<td>0.63(±0.05)</td>
</tr>
<tr>
<td>VPN Connections</td>
<td>85.55(±107)</td>
</tr>
<tr>
<td>VPN Gateway</td>
<td>2.52(±0.83)</td>
</tr>
<tr>
<td>Total Cost</td>
<td>133.60(±131.14)</td>
</tr>
</tbody>
</table>
AMI preparation is not acceptable, but we can reduce the overhead by
- overlapping execution and preparation time
- using IaaS Cloud in Japan.

<table>
<thead>
<tr>
<th>Resource Extension Overhead [s]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AMI Preparation [from Japan]</td>
<td>5322(±0.00)</td>
</tr>
<tr>
<td>AMI Preparation [inside Amazon]</td>
<td>304(±145)</td>
</tr>
<tr>
<td>VM Creation</td>
<td>147.58(±80.13)</td>
</tr>
<tr>
<td>SGE Installation</td>
<td>11.98(±7.54)</td>
</tr>
<tr>
<td>Total Cost [without AMI preparation]</td>
<td>159.96(±87.67)</td>
</tr>
</tbody>
</table>
This overhead is acceptable, because the execution time of one NLP task spends 50min.

<table>
<thead>
<tr>
<th>Resource Reduction Overhead [s]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AMI Deletion</td>
<td>23s(±0.10)</td>
</tr>
<tr>
<td>VM Shutdown</td>
<td>0.01(±0.00)</td>
</tr>
<tr>
<td>SGE Uninstallation</td>
<td>0.01(±0.00)</td>
</tr>
<tr>
<td>Total Cost</td>
<td>0.02(±0.00)</td>
</tr>
</tbody>
</table>
Performance of Resource Extension

- **Experimental Settings**
  - local: 8 SGE execution nodes, 4 cores (2.4GHz AMD) in each node.

<table>
<thead>
<tr>
<th>Workload</th>
<th>#Tasks</th>
<th>AET*¹ [s] on 2.4GHz</th>
<th>AET [s] on 2GHz</th>
<th>AET [s] on 3.25GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light</td>
<td>200</td>
<td>290.33 (±43.36)</td>
<td>276.30 (±73.87)</td>
<td>307.95 (±90.18)</td>
</tr>
<tr>
<td>Heavy</td>
<td>100</td>
<td>2564.90 (±266.05)</td>
<td>2689.50 (±233.79)</td>
<td>2923.6 (±1043.21)</td>
</tr>
</tbody>
</table>

*¹ AET stands for average execution times.

- Two kinds of NLP tasks
- Amazon EC2 resources
Heavy workloads in Scenario B:

- $45\text{min} \times \text{ceil}(100/32) \sim 180\text{min}$, deadline is $150\text{min}$

12 instances costs $12.20

Proposed mechanism guaranteed the QoS (8367s) with the minimum cost
Conclusions

- A hybrid and secure execution mechanism was proposed
  - Hybrid Resources Management
  - Resource Extension and Reduction Algorithms
  - Secure Communication with Amazon VPC

- Performance of the mechanism was evaluated
  - Overheads are acceptable
  - Proposed mechanism guaranteed the QoS of the NLP.

- Future works
  - To show the cost efficiency
  - Experimental studies with longer tasks
Thank you for your attention

Contact information:
Email: xsunsmile@gmail.com
HP: researchmap.jp/sunhao
Resource Extension Algorithm(1)

- Algorithms of Resource Extension

**Local Resources Timeline**
- Start
- Reschedule1
  - To meet deadline
- Reschedule2
- Deadline

**EC2 Resources Timeline**
- N tasks to ec2.q
- Create X ec2 instances
  - When performance down
  - Create Y ec2 instances
- O instance

- X instances
- X+Y instances
- O instance
$N_{t\_local} = \lfloor N_{localslots} T_{remain} / T_{local} \rfloor$

$N_{ec2slots} = \lfloor (N_{t\_now} - N_{t\_local}) T_{cpu\_capacity} / T_{remain} \rfloor$

$N_{t\_local}$ indicates the number of tasks that should be executed on local resources, $N_{ec2slots}$ indicates the number of Amazon EC2 VM cores for the rescheduled tasks.

$S_{cpu\_capacity} = \text{planEC2Resources}(N_{ec2slots}, \text{cpu\_capacity})$

$C_{ec2now} = \text{calculateCost}(S_{cpu\_capacity})$

$T_{remain}$ is more than 2 times of $T_{cpu\_capacity}$:

$\text{planEC2Resources}(11, "2ECU") = [\text{m1.xlarge}*2, \text{m1.large}*2]$  
Otherwise:

$\text{planEC2Resources}(11, "2ECU") = [\text{m1.xlarge}*3]$
Resource Reduction Algorithm

```
loop
    // Detect if any job finished
    J = waitAnySGETasks()
    if J is a task running on Amazon EC2 resource then
        // 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)
        else
            deleteTask(J, R_J)
        end if
    end if
end loop
```
Heavy workloads in Scenario A:

- **Proposed Method**: Proposed mechanism guaranteed the QoS
- **Waiting**
- **Proposed Method**
- **Local Only**

7 instances, costs $5.44.

m1.xlarge + 6 m1.large
# Trace of Resource Extension

<table>
<thead>
<tr>
<th>Time Remain [s]</th>
<th>#RJ*¹/cores (local or ec2)</th>
<th>#JCF*² (#Tasks, Resources)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6620</td>
<td>98/30 (local)</td>
<td>85</td>
</tr>
<tr>
<td>6555</td>
<td>85/29 (local)</td>
<td>81</td>
</tr>
<tr>
<td>6494</td>
<td>79/27 (local)</td>
<td>73</td>
</tr>
<tr>
<td>6431</td>
<td>72/26 (local)</td>
<td>69</td>
</tr>
<tr>
<td>6309</td>
<td>62/20 (local)</td>
<td>49</td>
</tr>
<tr>
<td>6244</td>
<td>43/15 (local)</td>
<td>36</td>
</tr>
<tr>
<td>6180</td>
<td>32/12 (local)</td>
<td>28</td>
</tr>
<tr>
<td>5515</td>
<td>48/22 (ec2)</td>
<td>44</td>
</tr>
<tr>
<td>5394</td>
<td>48/24 (ec2)</td>
<td>47</td>
</tr>
<tr>
<td>4792</td>
<td>46/26 (ec2)</td>
<td>45</td>
</tr>
<tr>
<td>4096</td>
<td>18/12 (local)</td>
<td>17</td>
</tr>
</tbody>
</table>

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