Urgent Computing with Virtual Private Clusters (VPC)

Satoshi Matsuoka
(joint work with T. Endo, N. Maruyama, H. Sato, H. Jitsumoto, H. Nishimura, M. Tatezono, S. Yamasaki, ...)

Tokyo Institute of Technology
National Institute of informatics

The TSUBAME Production “Supercomputing Grid Cluster” Spring 2006-2010

Voltaire ISR9288 Infiniband 10Gbps
x2 (DDR next ver.)
~1310+50 Ports
~13.5Terabits/s (3Tbits bisection)

Sun Galaxy 4 (Opteron Dual core 8-socket)
10512core/657Nodes
21.7Terabytes
50.6TeraFlops

OS Linux (SuSE 9)
NAREGI Grid MW

Unified IB network

“Fastest Supercomputer in Asia” 9th on the 28th Top500@47.38TF

Managed via SGE
NAREGI MW (β2)
~100mil CPU hours

Storage
1.5PB 1.6 Petabyte (Sun “Thumper”)
0.1Petabyte (NEC iStore)

Lustre FS, NFS, CIF, WebDAV (over IP)
50GB/s aggregate I/O BW

ClearSpeed CSX600
SIMD accelerator
360 boards,
35TeraFlops(Current)

NEC SX-8i
(for porting)

10Gbps+External Network

70GB/s
Next Generation Grid “Everybody Supercomputes”---for Interactive, Urgent Computing

Next Generation Grid will give desktop extension experience, a.k.a. Web 2.0

- Different usage env.
- No HP sharing with client’s PC
- Special HW/SW, lack ISV support
- Lack of common development env. (e.g. Visual Studio)
- Simple batch based, no interactive usage/UI
- Grids are not that different (sometimes worse complexity)

Hmm, it’s like my personal machine

10,000 users
1,300 HPC users

“Everybody’s Supercomputer”

Seamless, Ubiquitous access and usage
The next generation "true" Grid

- Uniform execution environment per user/applications, irrespective of underlying resource configurations
  - “Everything Virtualized”----operating system, libraries, tools, applications, network directory configurations, ...
  - Possible now with “commodity” virtualization
  - But we don't quite have the whole technologies...

- QoS guarantees for interactive and urgent computing (throughout the system)
  - Intelligent mono- and multi-site allocations of multiple (virtualized) jobs

- Look and feel of desktop extensions, ubiquity (Web 2.0 and beyond)
Grids as (large) Virtual Private Clusters

What **users** apps/users will benefit from VPC?
- Seasoned, batch-oriented research codes: **less benefit**
- Interactive usage, urgent usage: **big benefit**
- Industry collaboration usage: **big benefit (sandbox)**
- Broadens/incubates supercomputer population

What is the **size** of the apps that will benefit?
- Small resource usage: **less benefit**
  - Dedicated resource easy
  - High probabilistic availability with Large shared resources (possibly distributed)
- Large (sets of) resource usage (e.g., large MPI program): **big benefit (in the 1000s)**
  - Includes whole bunch of low-parallel ensemble runs
TSUBAME in Production
Oct.1 2006 (phase 3) ~10400 CPUs

>10,000 User Registrations
>1300 Registered SC Users
Batch Queue Prediction on TSUBAME (work w/Rich Wolski, USCB)

- Long wait times for small jobs due to massive parameter sweep
- Long wait times for large jobs due to long-running MPI jobs that are difficult to pre-empt, and require apps-specific QoS (e.g., memory)
### Status as of Mar 13\(^{th}\), 2007

<table>
<thead>
<tr>
<th>QUEUE</th>
<th>FREE NODE</th>
<th>FREE CPU</th>
<th>FREE MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL</td>
<td>260</td>
<td>1676 CPU</td>
<td>5240 GB</td>
</tr>
<tr>
<td>- bes1(param 1920CPUs)</td>
<td>107</td>
<td>783 CPU</td>
<td>2786 GB</td>
</tr>
<tr>
<td>- bes2(param 1920CPUs)</td>
<td>110</td>
<td>606 CPU</td>
<td>1526 GB</td>
</tr>
<tr>
<td>- default</td>
<td>26</td>
<td>112 CPU</td>
<td>412 GB</td>
</tr>
<tr>
<td>- gridMathem</td>
<td>8</td>
<td>128 CPU</td>
<td>256 GB</td>
</tr>
<tr>
<td>- high</td>
<td>9</td>
<td>47 CPU</td>
<td>260 GB</td>
</tr>
<tr>
<td>- sla1(hpc 1920CPUs)</td>
<td>0</td>
<td>0 CPU</td>
<td>0 GB</td>
</tr>
<tr>
<td>- sla2(hpc 1920CPUs)</td>
<td>0</td>
<td>0 CPU</td>
<td>0 GB</td>
</tr>
</tbody>
</table>

~80% node utilization

~50% CPU utilization

=> Memory bound (despite 21.4 Tbytes of memory for 50.4Teraflops, ~x2 Earth Simulator)
Grid Resource Sharing with Virtual Clusters

- **Virtual Cluster**
  - Virtual Machines (VM) as computing nodes
    - Per-user customization of exec environment
    - Hides software heterogeneity
    - Seamless integration with user’s own resources
  - Interconnected via overlay networks
    - Hides network asymmetry
    - Overcomes private networks and firewalls
Multi-Site Virtual Cluster

- Must coordinate multiple large virtualized resources across sites
  - Incl. virtualized networks, storage
- Need high QoS guarantees
  - In seconds instead of hours
  - High reliability/Rapid recovery
- Don't have technologies in place
Our work on Middleware for VPC

- Assumes large interactive use will benefit
  - Rapid, scalable deployment, in the seconds / and in 1000s of nodes (CCGrid 2007)
  - Scheduling of deployment of multiple cross-site MPI job executions & process migration (XHPC 2006)
  - Efficient cross-site collective communication in the 1000s (CCGrid 2007)
  - Automated rapid recovery of MPI in the 1000s (IPDPS-DPDNS 2007)
  - Robust dynamic failure detection and performance monitoring in the 1000s (SC2006)
  - Automated, rapid, large-scale, data replication and caching in Tera/Petabytes (Grid 2006)
  - Performance model for VPC---installation, execution, migration in the above context (in the works)

- GOAL: TSUBAME 2.0 will be entirely virtualized
Overview of the VPC: Our Fast Virtual Cluster Installer

Construction Request → Construction Server → Virtual Cluster

Resource Selection

Virtual Cluster Requirement

User

Transfer VM disk image

Customize the environment

Site A

Boot VM

Site B

Virtual Cluster

Construction Server

Request

Resource Selection

User

Virtual Cluster Requirement

Transfer VM disk image

Customize the environment

Site A

Boot VM

Site B

Virtual Cluster

Construction Server

Request

Resource Selection

User

Virtual Cluster Requirement

Transfer VM disk image

Customize the environment

Site A

Boot VM

Site B

Virtual Cluster

Construction Server

Request

Resource Selection

User

Virtual Cluster Requirement

Transfer VM disk image

Customize the environment

Site A

Boot VM

Site B

Virtual Cluster

Construction Server

Request

Resource Selection

User

Virtual Cluster Requirement

Transfer VM disk image

Customize the environment

Site A

Boot VM

Site B

Virtual Cluster

Construction Server

Request

Resource Selection

User

Virtual Cluster Requirement

Transfer VM disk image

Customize the environment
VPC Installer Requirements

- Autonomic Scheduling of VM Resources
- Installation Server
- User
- Virtual Cluster Requirement
- Scalable image transfer
- Easy specification of installation request
- Fast environment construction on VM
- Site A
  - Virtual Cluster
  - VM Image
  - VM
  - Package
- Site B
  - Virtual Cluster
  - VM Image
  - VM
  - Package
Our Proposed Rapid/Scalable VPC Installation Technique

- **Fast** installation with cached virtual machine images *(in seconds, `fork()*)
- **Scalable** image transfer with a pipelined transfer technique and *intelligent caching* *(in the thousands)*
- **Flexible** per-user/app customization using an existing cluster management tool (currently LUCIE, NPACI ROCKS support planned)
Preliminary Breakdown of Virtual Cluster Installation Time on our Earlier System

Package installation is the most dominant factor.
Overview of our Package Caching Algorithm

- **Observation**
  - Caching every installed image NOT scalable
  - User's package request are not uniform with clustered tendencies

- **Our Caching Algorithm**
  - Cache only images with substantial commonality in packages that are frequently requested, from historical installation traces
    - Hierarchical cluster analysis effectively identifies commonality among packages
      - \{ [MPI make gcc], [condor blast], [MPI gcc], [condor java], ... \}
      - \{ [MPI gcc], [condor], ... \}
Clustering User Requests

Request 1. [A, B, E]  
35.3  1

Request 2. [A, B, C]  
46  1

Request 3. [B, D, E]  
37.3  1

Request 4. [A, B, D]  
41  1

Red: Package Size  
Blue: # of Occurrences

Greatest package commonality

Package | A | B | C | D | E
---|---|---|---|---|---
Size    | 13 | 13 | 20 | 15 | 9.3
Prioritizing Clusters

Request 1. [A, B, E]
\[
35.3 \times 1 = 35.3
\]

Request 2. [A, B, C]
\[
46 \times 1 = 46
\]

Request 3. [B, D, E]
\[
37.3 \times 1 = 37.3
\]

Request 4. [A, B, D]
\[
41 \times 1 = 41
\]

Priority \(=\) expected time reduction
\(= (\text{sizeof(Packages)}) \times \# \text{Occurrences}\)

<table>
<thead>
<tr>
<th>Package</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>13</td>
<td>13</td>
<td>20</td>
<td>15</td>
<td>9.3</td>
</tr>
</tbody>
</table>

Generate as a cache image
Overview

Package download

Site Manager
- Cluster Installer
- VM invoke

Request
- Package Repository
- Site Manager
- Creator Daemon
- Cache Manager
- Cache Generation

Cache Selection
- Cache Generation
- Cache Image

Node1
- VM invoke
- VM
- Cache image
- Base image
- Cache image
- Cache image

Node2
- VM
- Cache image
- Base image
- Cache image

Pipelined package transfer and automated installation

Pipelined cache image transfer
Overview of prototype implementation

- Xen as Virtual Machine Monitor [Barham et al. ’03]
- Lucie as base cluster installer [Takamiya et al. ’03]
  - Using Debian Linux and package management utility DPKG/APT
  - NPACI/SDSC ROCKS version being planned
- Two pipelined transfer subsystems
  - Switched according to data size
    - Dolly+ [Manabe ’01]---large images
    - MPICH bcast [Thakur et al. ’06]---relatively small images
## Experimental Evaluation

### Evaluation Metrics
- Virtual cluster construction time reductions
- Scalability with increasing number of VMs

### Evaluation environment

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>RAM</th>
<th>HDD</th>
<th>Network</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site Manager</td>
<td>Athlon2000+</td>
<td>1GB</td>
<td>IDE</td>
<td>Gigabit</td>
<td>Linux-2.6.12.6</td>
</tr>
<tr>
<td>Node Type 1</td>
<td>Opteron242</td>
<td>2GB</td>
<td>IDE</td>
<td>Ethernet</td>
<td>Linux-2.6.16-xen</td>
</tr>
<tr>
<td>Node Type 2</td>
<td>Opteron280</td>
<td>4GB</td>
<td>SATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node Type 3</td>
<td>Opteron250</td>
<td>2GB</td>
<td>SCSI</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th>Xen</th>
<th>Lucie</th>
<th>Dolly+</th>
<th>MPICH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3.0.2-2</td>
<td>0.0.5</td>
<td>0.93-1</td>
<td>1.2.7p1</td>
</tr>
</tbody>
</table>

*Note the heterogeneity of CPU, RAM, HDD*
Artificial Request Generation

- Still difficult to collect large set of real requests for benchmarking => generate artificial requests
- Assume request consists of common core packages with varying small extra packages
- 720 artificial package requests generated
  - Core: 16, Extra: 45 packages
Effect on VPC Construction Time

- Random installation requests for 50-VM virtual clusters with identical core package set (200 installations)
  - VM spec: 256MB memory, 2GB disk
  - Dolly+ as a pipelined transfer agent

- Generate a 5 GB cache image every 50 installations

![Graph showing the effect on VPC construction time with and without caching.](image)

- Average 42.3% install time reduction w/caching
- 27.2 sec. fastest

Cache generation

<table>
<thead>
<tr>
<th># installations</th>
<th>Package cover rate (%)</th>
<th>Construction time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>160</td>
</tr>
<tr>
<td>100</td>
<td>90</td>
<td>140</td>
</tr>
<tr>
<td>150</td>
<td>80</td>
<td>120</td>
</tr>
<tr>
<td>200</td>
<td>70</td>
<td>100</td>
</tr>
</tbody>
</table>
Breakdown of Installation Time

- **No cache**
  - Pre-processing
  - Package download
  - Package transfer
  - Package installation
  - Per node customization
  - Reboot

- **Use cache**
  - Reduced package installation time
  - Increased cache selection time

Construction time (sec)
Scalability w/# of VPC nodes

- Installation time of virtual clusters up to 200 VMs
  - 5.8MB package installed
  - VM spec: 256MB memory, 1GB disk

Graph:
- 23-VM cluster: approx. 30 secs
- 200-VM cluster: approx. 37 secs
- Install time bound by the slowest HDD? => under investigation
Scalability w/# of VPC nodes: Optimistic Extrapolation to 1000 VMs

Likely to be due to some unstable HDDs

1000-VM virtual cluster in less than 1 minute!

If we pruned unreasonably-slow HDDs…

If we pruned unreasonably-slow HDDs…

Whole  Transfer  Installation

Number of nodes

Construction time (sec)

0  5  10  15  20  25  30  35  40

0  200  400  600  800  1000
Existing virtual cluster construction system

- Virtual cluster
  - Virtual Machines (VM) as computing nodes
  - Customizable without interfering with underlying environments

More than 100 sec even in a single site

Difference of machine spec leads varied set up time

150 nodes
MPI, gcc
CPU > 2.0GHz

Difference of VM setup time between nodes

Site A

Site B
Steps in VM setup processes

1. Download Packages
2. Transfer Packages
3. Install Packages
4. Customize script exec.
5. Reboot

Cluster Installer
Head node
Package Repository
Required VM
Minimum image
Const. request
## Model of VM setup time in each steps

<table>
<thead>
<tr>
<th>Step</th>
<th>Process</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>Download Packages</td>
<td>$\alpha_1(PkgSize) + \beta_1$</td>
</tr>
<tr>
<td>Step 2</td>
<td>Transfer Packages</td>
<td>$\alpha_2(PkgSize) + \beta_2(TransferOrder) + \gamma_2$</td>
</tr>
<tr>
<td>Step 3</td>
<td>Install Packages</td>
<td>$\alpha_3(PkgSize) + \beta_3(CPUfreq)^{-1} + \gamma_3(DiskWrite)^{-1} + \delta_3$</td>
</tr>
<tr>
<td>Step 4</td>
<td>Customize Script exec.</td>
<td>$\alpha_4(PkgSize) + \beta_4(CPUfreq)^{-1} + \gamma_4(DiskWrite)^{-1} + \delta_4$</td>
</tr>
<tr>
<td>Step 5</td>
<td>Reboot</td>
<td>$\alpha_5(PkgSize) + \beta_5(CPUfreq)^{-1} + \gamma_5(DiskRead)^{-1} + \delta_5$</td>
</tr>
</tbody>
</table>

- $PkgSize$: Package size installed additionally (MB)
- $TransferOrder$: Package transfer order
- $CPUfreq$: CPU Frequency (GHz)
- $DiskRead$: Disk Read Speed (MB/s)
- $DiskWrite$: Disk Write Speed (MB/s)

**Decide coefficients by multiple regression analysis**
Experiments for deciding coefficients

Exp. datas

- When constructing virtual cluster with existing system [Nishimura et al. ’07]
  - About 200VMs

- Package size pattern
  - 31 kinds (0, 5, …, 30 MB)

Exp. env (Presto III test cluster, 260 nodes/600 CPUs)

<table>
<thead>
<tr>
<th>Composition</th>
<th>CPU</th>
<th>RAM</th>
<th>HDD</th>
<th>Network</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition 0</td>
<td>Athlon2000+</td>
<td>1GB</td>
<td>IDE</td>
<td>Gigabit Ethernet</td>
<td></td>
</tr>
<tr>
<td>Composition 1</td>
<td>Opteron242</td>
<td>2GB</td>
<td>IDE</td>
<td>Gigabit Ethernet</td>
<td></td>
</tr>
<tr>
<td>Composition 2</td>
<td>Opteron280</td>
<td>4GB</td>
<td>SATA</td>
<td>Gigabit Ethernet</td>
<td></td>
</tr>
<tr>
<td>Composition 3</td>
<td>Opteron250</td>
<td>2GB</td>
<td>SCSI</td>
<td>Gigabit Ethernet</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS</th>
<th>VMM</th>
<th>Installer</th>
<th>Data transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux-2.6.16</td>
<td>Xen 3.0.2-2</td>
<td>Lucie 0.0.5</td>
<td>Dolly+ 0.93-1</td>
</tr>
</tbody>
</table>
### Results

<table>
<thead>
<tr>
<th>Step Name</th>
<th>Model Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download Pkg</td>
<td>$0.312 \times \text{PkgSize} + 0.72$</td>
</tr>
<tr>
<td>Transfer Pkg</td>
<td>$0.022 \times \text{PkgSize} + 0.04 \times \text{TransferOrder} + 0.14$</td>
</tr>
<tr>
<td>Install Pkg</td>
<td>$0.784 \times \text{PkgSize} + 56 \times (\text{CPUfreq})^{-1} + 308 \times (\text{DiskWrite})^{-1} - 47$</td>
</tr>
<tr>
<td>Customize Script exec.</td>
<td>$0.016 \times \text{PkgSize} + 1.1 \times (\text{CPUfreq})^{-1} + 13 \times (\text{DiskWrite})^{-1} - 0.8$</td>
</tr>
<tr>
<td>Reboot</td>
<td>$0.645 \times \text{PkgSize} + 9.9 \times (\text{CPUfreq})^{-1} + 62 \times (\text{DiskRead})^{-1} + 6.7$</td>
</tr>
</tbody>
</table>

### MAX & MIN of PrestoIII machine spec

<table>
<thead>
<tr>
<th></th>
<th>MAX</th>
<th>MIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Frequency (GHz)</td>
<td>2.4</td>
<td>1.6</td>
</tr>
<tr>
<td>Disk read speed (MB/s)</td>
<td>69.3</td>
<td>18.4</td>
</tr>
<tr>
<td>Disk write speed (MB/s)</td>
<td>76.4</td>
<td>10.8</td>
</tr>
</tbody>
</table>
## Evaluation of model accuracy

<table>
<thead>
<tr>
<th>Step name</th>
<th>Coefficient of determination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download Packages</td>
<td>0.9959</td>
</tr>
<tr>
<td>Transfer Packages</td>
<td>0.9978</td>
</tr>
<tr>
<td>Install Packages</td>
<td>0.9236</td>
</tr>
<tr>
<td>Customize</td>
<td>0.5593</td>
</tr>
<tr>
<td>Script exec</td>
<td>0.468</td>
</tr>
</tbody>
</table>

**Figure:** Measurement and prediction of VM setup time

VM setup times of each node (when package size = 45 MB)

More than 0.9 precision
Simulation result

- Simulated PrestoIII cluster in our lab.
- Compared different resource selection policies

<table>
<thead>
<tr>
<th>Package size and # of consisting nodes</th>
<th>VM setup time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50MB/50VM</td>
<td>60</td>
</tr>
<tr>
<td>50MB/100VM</td>
<td>80</td>
</tr>
<tr>
<td>100MB/50VM</td>
<td>100</td>
</tr>
<tr>
<td>100MB/100VM</td>
<td>120</td>
</tr>
</tbody>
</table>

Model

- CPU: Reduced const. time by MAX 33.7 sec
- Disk: Disk policy is also better

For increasing package size and nodes, the difference is evident.

Diagram notes:
- VM setup time for different selection policies.
- Package size and # of consisting nodes.

Legend:
- Model
- CPU
- Disk
- FIFO
TSUBAME Scheduling and Accounting
--- Synonymity w/ Existing Social Infrastructures

- Three account/queue types (VO-based)
  - Small FREE Usage: "Promotion Trial (Catch-and-bait)"
  - Service Level Agreement: "Cell Phones"
    - Exclusivity and other high QoS guarantees
  - Best Effort (new): "Internet ISP"
    - Flat allocation fee per each "UNIT"
  - Investment Model for allocation (e.g. "Stocks & Bonds")
    - Open & extensive information, fair policy guarantee
    - Users make their own investment decisions---collective societal optimization (Adam Smith)

Q: Social infrastructure analogy for Urgent Computing?

Dynamic machine-level resource allocation
SLA > BES > Small

10,000 accounts
Over 1300 SC users

64CPUs
64CPUs
64CPUs
64CPUs
64CPUs
64CPUs

Jan
Feb
Mar

Nano-VO
Max CPU=192
Future Work

- More elaborate study of performances and the bottlenecks
- Model-based autonomous node selection for efficient and robust virtual cluster deployment
  - Automatically prune less reliable nodes
  - Optimal matchmaking of jobs and resources
    - CPU intensive: allocate nodes with faster CPUs
    - Network intensive: deploy onto nodes with faster interconnects
- SDSC/NPACI ROCKS as installer
  - Collaboration started with ROCKS group
- Next Gen Petascale TSUBAME 2.0 in 2010 WILL be fully virtualized