Enabling Urgent Computing within the Existing Distributed Computing Infrastructure

Ph.D. Dissertation Defense
Computer Science

Nick Trebon
6/20/11
University of Chicago
Committee

- Pete Beckman (ANL)
- Ian Foster
- Rick Stevens
Acknowledgements

- SPRUCE:
  - Pete Beckman (PI), Suman Nadella, Ivan Beschastnikh, Jason Cope, Dylan Stark

- Rich Wolski and Dan Nurmi, University of California at Santa Barbara

- TeraGrid
Dissertation Map

- Experimental vs. theoretical
- Approach
  - Formulate question
  - Build and deploy system to explore associated problem space
  - Gather data
  - Analyze results
  - Generate conclusions
Outline

• Introduction
  - Definition and motivation
  - Problem statement
• SPRUCE: Urgent Computing Framework
• Resource Selection
• Clouds as Urgent Computing Resources
• Contributions, Conclusions and Future Work
Introduction: Distributed Computing

- Distributed and high performance computing resources
  - Provide users with multiple compute elements
  - Types: clusters, supercomputers, clouds, Grids, condor pools, etc.
  - Resources shared by large community of users
  - How to deal with resource contention?
    - Reject, queue, make space available
  - Batch queue is popular choice
    - Queue delay is highly variable and may be greater than execution delay (Wolski)
Urgent Computing

• **Definition:** An *urgent computation* is one that:
  - 1. Has a deadline (i.e. critical decision making window)
  - 2. Has an unpredictable onset
  - 3. Requires significant resource usage

• Examples: Severe weather modeling, wildfire prediction, urban airflow modeling, etc.
How is Urgent Computing Done Now?

• User requests elevated priority from a resource administrator
• If priority access is granted, administrator is responsible for scheduling urgent computations on to resource
  - The administrator is a single point of failure
• The urgent computing group may be interested in multiple resources that span multiple administrative domains
• Example: LSU hurricane center requesting allocation on ANL resources in 2008
Problem Statement

• What combination of management mechanisms, policies, and tools is needed in order to support urgent computing on supercomputers, clusters and computational clouds?
Preview: Contributions

• Define *urgent computing*
• Identify and evaluate set of statistical methods for probabilistic bounds prediction
• Propose and evaluate elevated priority policies for urgent computing resources
• Technical contribution: prototype framework
Related Work (1/2)

• Real-Time computing
  - Overall correctness = result correctness + time results produced
  - Examples: flight control systems, robotics, automotive applications, etc.
  - Hard Real-Time vs. Soft Real-Time
    - Hard: missing deadline could have *catastrophic* consequences to environment
    - Soft: meeting deadline preferred, but no adverse affect on environment
    - Urgent computing lands in between hard and soft real-time computing
  - Real-time systems are typically complete systems designed specifically for real-time application
  - Urgent computing operates at larger order of magnitude
Related Work (2/2)

• Grid resource selection
  • Shares many traits in common with urgent computing resource selection
    ▪ Multiple resources that belong to distinct administrative domains
    ▪ Distinct software environments and hardware
  • Most of the research focuses on minimizing total turnaround time (or some aspect)
    ▪ Urgent computing interested in deadline feasibility
  • Urgent computing resource selection must take into account multiple elevated priority policies supported at each resource
Outline

• Introduction
  ♦ Definition and motivation
  ♦ Problem statement

• SPRUCE: Urgent Computing Framework

• Resource Selection

• Clouds as Urgent Computing Resources

• Contributions, Conclusions and Future Work
SPRUCE

• Special Priority and Urgent Computing Environment (SPRUCE)
  ✦ An open-source, token-based framework to manage:
    ▪ Users (token manager, team member)
    ▪ Resources (batch, clouds, network)
    ▪ Policies
    ▪ Sessions
  ✦ Purpose: Provide fast and efficient priority access for urgent computations while
    ▪ Resources maintain autonomy
    ▪ Eliminating single point of failures
Outline

• Introduction
  - Definition and motivation
  - Problem statement
• SPRUCE: Urgent Computing Framework
• Resource Selection
• Clouds as Computational Resources
• Contributions, Conclusions and Future Work
Resource Selection

- Variation of grid resource selection problem
  - Deadline feasibility vs. minimization
    - How to *configure* workflow (resource, policy, data repositories, runtime parameters) to meet deadline?
- Approach: Calculate probabilistic upper bound for computation
- What if no configuration can meet deadline?
  - Scale down problem size
Total Turnaround Time

- **Input Phase**: $(I_Q, I_B)$
- **Resource Allocation Phase**: $(R_Q, R_B)$
- **Execution Phase**: $(E_Q, E_B)$
- **Output Phase**: $(O_Q, O_B)$

- If each phase is independent, then:
  - Overall bound $= I_B + R_B + E_B + O_B$
  - Overall quantile $\geq I_Q \times R_Q \times E_Q \times O_Q$
- Improving the bounds
  - Speculative execution
  - Overlapping phases
Resource Selection Case Study

• Application: flash2
• File staging requirements:
  - Modeled after LEAD workflow
  - Input/Output file: 3.9 GB
  - Repository: Indiana (input) and UCAR (output)

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>CPU</th>
<th>Total Nodes</th>
<th>Total CPUs</th>
<th>Req. Nodes</th>
<th>Req. PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC/ANL–IA64</td>
<td>Itanium 2 (1.3/1.5 GHz)</td>
<td>62</td>
<td>124</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Mercury</td>
<td>Itanium 2 (1.5 GHz)</td>
<td>631</td>
<td>1262</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>Tungsten</td>
<td>Intel Xeon (3.5 GHz)</td>
<td>1280</td>
<td>2560</td>
<td>32</td>
<td>64</td>
</tr>
</tbody>
</table>
Default Priority Experiments

Results Table
Default Priority Experiments

Results Table
Elevated Priority Experiments

Results Table
Resource Selection Conclusions

• Overall, individual and composite bounds perform well for elevated priority policies
• Elevated priority polices = less variability and greater predictability
• TCP–based probes unable to predict GridFTP bandwidth
Outline

• Introduction
  - Definition and motivation
  - Problem statement
• SPRUCE: Urgent Computing Framework
• Resource Selection
• Clouds as Urgent Computing Resources
• Contributions, Conclusions and Future Work
Introduction to Clouds

- Infrastructure-as-a-Service
  - Hardware/location hidden from user

- Virtualization
  - Hypervisor: Physical resource manager
  - Virtual machines: user-defined software implementation of a machine
    - Multiple independent instances per image
    - Image portability
    - Single system can host multiple VMs

- Examples: Amazon’s EC2, Magellan
Urgent Computing with Clouds

• Target finite-capacity clouds
  - Example: ANL’s Magellan
  - Type of cloud hosted by research facility (open-source provisioner)
  - No batch queue

• Urgent computing support: extend illusion of infinite capacity

• Cloud allocation phase:
  - Image transfer → instance preparation → instance startup
  - Time to start all requested virtual machines
# Evaluated Cloud Policies

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preemption</td>
<td>Nonurgent VMs are terminated.</td>
</tr>
<tr>
<td>Suspension</td>
<td>Nonurgent VMs are suspended and their memory contents are written to disk. CPU and memory resources are then released. Suspended virtual machines may be restarted once resources are again available.</td>
</tr>
<tr>
<td>Migration</td>
<td>Smaller nonurgent VMs are migrated to create the necessary room for urgent VMs.</td>
</tr>
<tr>
<td>VM QoS</td>
<td>The number of virtual CPUs or amount of memory allocated to nonurgent VMs is reduced to free those resources for urgent VMs.</td>
</tr>
</tbody>
</table>
# Cloud Case Study Details

## Cloud Hardware Details:

<table>
<thead>
<tr>
<th></th>
<th>Breadboard Clouds</th>
<th>Magellan Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>24</td>
<td>504</td>
</tr>
<tr>
<td>Memory (GB)</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>Disk (GB)</td>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>Cores</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

## Virtual Machine Types:

<table>
<thead>
<tr>
<th>VM Type</th>
<th>Breadboard Clouds</th>
<th>Magellan Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mem (MB)</td>
<td>Disk (GB)</td>
</tr>
<tr>
<td>m1.small</td>
<td>128</td>
<td>2</td>
</tr>
<tr>
<td>c1.medium</td>
<td>256</td>
<td>6</td>
</tr>
<tr>
<td>m1.large</td>
<td>512</td>
<td>10</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>1,024</td>
<td>20</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>2,048</td>
<td>20</td>
</tr>
</tbody>
</table>
Case Study Methods

• Clouds:
  - bb-prod (default Eucalyptus 1.6.2)
  - bb-dev (Eucalyptus–1.6.2 + priorities)
  - Magellan (Eucalyptus–1.6.2)

• Images:
  - 1 GB CentOS image (target)
  - 11 GB MG–RAST image (urgent)
    ▪ Required largest instance type (c1.xlarge)

• Experiments:
  - 100 trials, measure allocation delay
  - Images precached on nodes
Single Node VM Allocation

- Purpose: determine how VM type and number of instances per node affects delay
- bb-prod cloud
- 1 GB image
  - precached
- Size of disk is key attribute
- Results table
Single Node VM Allocation with Empty Cache

- 11 GB MG–RAST image
  - c1.xlarge instance type
- Cache: 2 instances → 2 disk copies
- No Cache: 1 instance → 1 disk copy + 1 network transfer

Average and standard deviations (in seconds) for measured allocation delay:

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>Cache</th>
<th>No Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>μ</td>
<td>σ</td>
</tr>
<tr>
<td>1</td>
<td>331</td>
<td>21.2</td>
</tr>
<tr>
<td>2</td>
<td>527</td>
<td>45.9</td>
</tr>
</tbody>
</table>
Multiple Node VM Allocation

• Purpose: determine effect of increasing request size on allocation delay
  ✤ Instances per node manipulated
• bb-prod cloud
• 11 GB MG-RAST image
  ✤ c1.xlarge instance type
  ✤ precached
• Results table
Breadboard Cloud Conclusions

• Unexpected scaling behavior
  - Influenced by individual differences in node performance
  - Allocation process modified to use sparse files for ephemeral disk
    - Shorter allocation delays, but little impact on scaling behavior --- >8 nodes causes problems

• Primary factors of delay
  - Image exists in cache
  - Number of instances per node
  - VM Type (size of disk attribute)
Magellan Provisioner Comparison

- **Eucalyptus 1.6.2**
  - Poor scaling behavior, stability

- **Eucalyptus 2.0**
  - Improved stability and scalability

- **OpenStack**
  - Fast allocation delays, poor scaling (currently under investigation by admins)
Elevated Priority Experiments

- bb-dev cloud
- Target instances: 1 GB CentOS image
- Urgent instances: 11 GB MG–RAST image
- All images precached
- Worst-case cloud state
  - Cloud filled to capacity with target instances
- Scheduler: minimize number of urgent instances per node

Migration Details

Preliminary QoS Experiment
Comparison: Single Node Elevated Priority

- Breadboard (Eucalyptus 1.6.2)
- Target instances: 1-GB CentOS,
- Urgent instances: MGRAST image, c1.xlarge instance type

Preemption  Suspension
Migration    VM QoS
Comparison: Multiple Node Elevated Priority

- Breadboard (Eucalyptus 1.6.2)
- Target type: c1.xlarge
- Problem with 2–instance default delays on more than 4 nodes

Preemption  Suspension  Migration  VM QoS
Predicting Bounds for VM Allocation Delay

- Setup: 8–node cloud on Breadboard
- Prediction method: modified BMBP to predict 0.95 quantile
- 100 trials to seed, 100 trials to evaluate

<table>
<thead>
<tr>
<th>Policy</th>
<th>1 Instance per Node</th>
<th>2 Instances per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Correctness</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Default</td>
<td>92%</td>
<td>4%</td>
</tr>
<tr>
<td>Preemption</td>
<td>100%</td>
<td>11%</td>
</tr>
<tr>
<td>Suspension</td>
<td>92%</td>
<td>9%</td>
</tr>
<tr>
<td>VM QoS</td>
<td>87%</td>
<td>8%</td>
</tr>
</tbody>
</table>
Predicting Bounds for Preemption Allocation Delay

Graphs showing delay bounds for 8 and 16 instances over trials.
Cloud Experiment Conclusions

• Improvements to urgent VM allocation algorithm
  - Sparse files, copy–on–write format (OpenStack)
  - Precaching certain urgent images
  - Optional caching (Eucalyptus)

• Elevated priority policies
  - Preemption, suspension VM QoS
    - Little overhead
    - Minimal delay savings in targeting smaller VM types
  - Migration
    - Expensive
    - Target small VMs
    - Potentially useful to clouds in general
Outline

• Introduction
  - Definition and motivation
  - Problem statement
• SPRUCE: Urgent Computing Framework
• Resource Selection
• Clouds as Urgent Computing Resources
• Contributions, Conclusions and Future Work
Contributions (1/2)

- Formally defines and explores problem-space of *urgent computing*
- Technical contribution: prototype framework
  - Manages users, resources, policies and sessions
  - Supports supercomputers, clusters, grids, condor pools and clouds
Contributions (2/2)

- Identifies and analyzes set of heuristics to predict probabilistic bounds
  - Inability of TCP-probes to predict GridFTP bandwidth
  - First known research to utilize next-to-run and modified BMBP methods in resource selection context
- Evaluates urgent computing policies for batch and cloud computational resources
Urgent Computing Conclusions

- Resources that support urgent computing should:
  - Provide elevated priority access policies
  - Educate and inform all users
  - Offer incentives/compensation to users who have running jobs affected
    - Examples: one-time use token, lower charge rate

- Elevated priority policies should seek to:
  - Reduce average allocation delay
  - Provide greater predictability
VM Allocation
Conclusions (1/2)

• Use sparse files and copy-on-write formats to manage disk resources
• Allow urgent users to skip image caching in order reduce allocation delay
• Precache urgent image during period of higher frequency (e.g., hurricane season)
• A cache-aware VM scheduler could target nodes where the urgent image is cached
VM Allocation
Conclusions (2/2)

• Clouds with shared storage between nodes
  - Nodes could share an image cache, reducing number of images to be downloaded from image repository
  - Nodes could support live migration
    - Clouds could automatically relocate images when an undesirable cloud state is detected
    - Poorly behaving nodes could have instance migrated and be taken offline for diagnosis
    - Idle VMs relocated to oversubscribed nodes until they become active again
Cloud Conclusions

• Nonurgent users should be able to designate VMs as urgent computing targets
  - Graceful degradation of services (e.g., replica servers)

• Instances specify minimum memory and CPU requirements
  - Improve safety of QoS policy
  - VM QoS
Open Topics for Future Explorations

• Network bandwidth as a resource
  ✤ Reserve guaranteed bandwidth between end hosts

• Expand cloud experiments
  ✤ Implement policies in OpenStack
  ✤ Evaluate policies on larger clouds
  ✤ Evaluate scalability of migration policy

• Evaluate new cloud architectures
  ✤ Shared storage between nodes
  ✤ Intel’s 48-core single-chip cloud computer

• Refine probabilistic bounding methods
  ✤ Account for presence of other urgent computing jobs
Questions ?
Extra slides
CHAPTER 1: INTRODUCTION
Urgent Computing Infrastructure Requirements

• Management mechanisms
  ✷ User Management
  ✷ Resource Management
  ✷ Session Management for priority access

• Elevated priority resource policies
  ✷ Set of policies for each resource type

• Tools
  ✷ Resource selection: which resources and policies provide best probability of meeting deadline?
Related Work (1/3)

- On-demand computing
  - Commonly referred to as “utility computing”
  - Consumers lease computational infrastructure as needed
  - Ability to add/subtract capacity on the fly
  - Often uses virtualization
  - Example: Clouds (e.g., Amazon EC2, Eucalyptus Public Cloud, Nimbus, etc.)
  - Papers: “EC2 currently not suitable for HPC”
    - HPC = tightly coupled applications with significant memory and communication requirements
CHAPTER 2: SPRUCE DETAILS
Batch Computational Resources

- Examples: Supercomputers, clusters, grids
- Suitable for HPC and distributed applications
- Resource managers (e.g., LoadLever, PBS, etc.)
- Potential elevated priority policies:
  - Elevated priority
  - Next-to-run
  - Preemption
Cloud Resources

• Infrastructure-as-a-Service (IAAS)
  ♦ Examples: EC2, Magellan

• Benefits to urgent computing users
  ♦ On-demand nature
  ♦ Virtual machines provide flexibility

• “Infinite capacity” vs. “finite capacity”
  ♦ Large commercial clouds are specifically designed to be overprovisioned

• Target smaller, “finite capacity” clouds
  ♦ What happens when cloud is at capacity?
    ▪ Goal: provide illusion of “infinite capacity” to urgent users
Network Bandwidth

• Urgent applications may have significant file staging requirements
  - Real-time (or latest) sensor data to be transferred to targeted computational resource

• Goal: incorporate and evaluate systems into urgent computing framework that provide:
  - Guaranteed bandwidth
  - Higher average achievable bandwidth
  - Greater predictability
  - Examples: NLR’s Dynamic VLAN Service, OSCARS

• (Future Work)
# Resource Policies

<table>
<thead>
<tr>
<th>Computational Resources</th>
<th>Network Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td>Dynamic VLAN</td>
</tr>
<tr>
<td>Elevated Priority</td>
<td>Urgent Access Only</td>
</tr>
<tr>
<td>Next-to-run Preemption</td>
<td>VM QoS Migration</td>
</tr>
<tr>
<td></td>
<td>Suspension</td>
</tr>
<tr>
<td></td>
<td>Preemption</td>
</tr>
<tr>
<td>Condor Pools</td>
<td></td>
</tr>
<tr>
<td>Elevated Priority</td>
<td></td>
</tr>
<tr>
<td>Preemption</td>
<td></td>
</tr>
</tbody>
</table>

- Dissertation Scope: Batch and cloud resources
SPRUCE Sessions

- Agreements between users and resource providers
- All attributes (except users) agreed to ahead of time
- Resource administrators maintain autonomy
Technical Details

Web Portal or Workflow Tools
- AJAX
- PHP / Perl

Client Interfaces
- SOAP Request

Computing Resource:
- Job Manager & Scripts
  - Java
  - Axis2
  - PHP / Perl

Client-Side Job Tools
- SOAP Request

Central SPRUCE Server
- SPRUCE User Services
- Validation Services
  - Axis 2 Web Service Stack
  - Tomcat Java Servlet Container
  - Apache Web Server

MySQL
- JDBC

Future work

Back
Token Manager Workflow
Team Member Workflow

User Team

Urgent Computing Job Submission

Conventional Job Submission Parameters

Urgent Computing Parameters

Choose a Resource

SPRUCE Job Manager

Authentication

Priority Job Queue

Local Site Policies

Back
CHAPTER 3: RESOURCE SELECTION
**I₇B+R₇B+E₇B+O₇B**: File Staging Delay

- Method is the same for input/output file staging
- Utilize Network Weather Service to generate bandwidth predictions based upon probe
  - Wolski, et al.: Use MSE as sample variance of a normal distribution to generate upper confidence interval
    - E.g., upper 95% = $\text{forecast} \times 1.64 \times \sqrt{\text{MSE}}$
- Challenges
  - Predicting bandwidth for output file transfers
  - NWS uses TCP-based probes, transfers utilize GridFTP

Back
TCP Probes vs. Large GridFTP Transfers

Left: 1,049 2_Mb probes between University of Indiana input to NCSA resource
Right: 1,049 171-MB GridFTP transfers between same end hosts
TCP Probes vs. GridFTP Transfers

Left: 154 2-MB probes between University of Indiana input to NCSA resource
Right: 154 2-MB GridFTP transfers between same end hosts
GridFTP Probe Framework

- **Purpose:** to generate probe measurements whose behavior matches the behavior seen in large GridFTP transfers
- **Probes:**
  - Tuned for each source, destination pair
  - Sent every 15 minutes
  - Initiated as third-party transfers
  - Measurements stored directly into NWS
- **Results:**
  - More similarity between probe and transfer behavior
  - Correctness and accuracy of predicted bounds discussed later
IB + RB + EB + OB: Resource Allocation Delay

- **Batch resources**
  - Default priority (no SPRUCE)
    - Queue Bounds Estimation from Time Series (QBETS)
  - Elevated priority policies
    - Next-to-run
      - Modified QBETS – Monte Carlo simulation on previously observed job history
    - Preemption
      - Utilize modified Binomial Method Batch Predictor (BMPB) on preemption history

- **Concerns**
  - History vs. current queue state
  - Multiple instances
I_B + R_B + E_B + O_B: Execution Delay

- Approach: Generate empirical bound for a given urgent application on each target resource
  - Set of potential resources is known ahead of time
  - Utilize modified BMBP methodology to generate probabilistic bounds
    - Requires execution history
    - Methodology is general and non-parametric
Improving the Bounds

• Composite quantile quickly decreases
  - $0.95^4 = 0.815$
  - $0.75^4 = 0.316$

• Speculative execution
  - Initiate two *independent* configurations
  - $\Pr(C_1 \text{ or } C_2) = \Pr(C_1) + \Pr(C_2)(1 - \Pr(C_1))$

• Overlapping phases
  - E.g., stage input files while job is queued
Overlapping Phases Example

- Input file staging
  - \( \Pr(10 \text{ minutes}) = 0.90 \)
  - \( \Pr(20 \text{ minutes}) = 0.95 \)
  - \( \Pr(30 \text{ minutes}) = 0.99 \)

- Queue delay
  - \( \Pr(10 \text{ minutes}) = 0.75 \)
  - \( \Pr(20 \text{ minutes}) = 0.85 \)
  - \( \Pr(30 \text{ minutes}) = 0.95 \)

- Serial: \( \Pr(\text{job begins in} < 50 \text{ minutes}) = 0.95 \times 0.95 = 0.9 \)

- Overlap: \( \Pr(\text{job begins} < 30 \text{ minutes}) = 0.95 \times 0.99 = 0.94 \)
Approximating Configuration Probability

- Goal: to determine a probabilistic upper bound for a given configuration

- Approximate approach: query a small subset (e.g., 5) of quantiles for each individual phase and select the combination that results in highest composite quantile with bound < deadline
Limitations of Bounds

• Assume workflow runs on single resource
• Bounds correspond to single instance
  ♦ 1 instance vs. 1,000 — bounds do not account for correlation
• Resource allocation bounds
  ♦ Do not account for presence of SPRUCE jobs
  ♦ Based on past history, not on current state
# Default Priority Phase Results

## Correctness:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Number of Trials</th>
<th>Input</th>
<th>Queue</th>
<th>Execution</th>
<th>Output</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC/ANL</td>
<td>99</td>
<td>100%</td>
<td>97%</td>
<td>100%</td>
<td>83%</td>
<td>97%</td>
</tr>
<tr>
<td>Mercury</td>
<td>100</td>
<td>99%</td>
<td>99%</td>
<td>93%</td>
<td>86%</td>
<td>100%</td>
</tr>
<tr>
<td>Tungsten</td>
<td>100</td>
<td>88%</td>
<td>99%</td>
<td>96%</td>
<td>77%</td>
<td>99%</td>
</tr>
</tbody>
</table>

## Accuracy:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Input</th>
<th>Queue</th>
<th>Execution</th>
<th>Output</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC/ANL</td>
<td>7.98%</td>
<td>2,379.09%</td>
<td>7.01%</td>
<td>21.89%</td>
<td>64.95%</td>
</tr>
<tr>
<td>Mercury</td>
<td>10.19%</td>
<td>4,468.23%</td>
<td>3.78%</td>
<td>5.29%</td>
<td>1,417.26%</td>
</tr>
<tr>
<td>Tungsten</td>
<td>10.57%</td>
<td>4,811.20%</td>
<td>13.03%</td>
<td>-4.63%</td>
<td>1,842.43%</td>
</tr>
</tbody>
</table>

## Summary (in seconds):

<table>
<thead>
<tr>
<th>Resource</th>
<th>Average Delay</th>
<th>Average Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC/ANL</td>
<td>10,833</td>
<td>17,869</td>
</tr>
<tr>
<td>Mercury</td>
<td>8,696</td>
<td>131,951</td>
</tr>
<tr>
<td>Tungsten</td>
<td>7,631</td>
<td>148,233</td>
</tr>
</tbody>
</table>
Elevated Priority Phase Results

Correctness:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Number of Trials</th>
<th>Input</th>
<th>Queue</th>
<th>Execution</th>
<th>Output</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next-to-Run Preemption</td>
<td>98</td>
<td>100%</td>
<td>92%</td>
<td>91%</td>
<td>92%</td>
<td>94%</td>
</tr>
<tr>
<td></td>
<td>109</td>
<td>98%</td>
<td>94%</td>
<td>93%</td>
<td>85%</td>
<td>95%</td>
</tr>
</tbody>
</table>

Accuracy:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Input</th>
<th>Queue</th>
<th>Execution</th>
<th>Output</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next-to-Run</td>
<td>8.04%</td>
<td>46.37%</td>
<td>4.54%</td>
<td>41.47%</td>
<td>11.27%</td>
</tr>
<tr>
<td>Preemption</td>
<td>5.72%</td>
<td>75.08%</td>
<td>4.39%</td>
<td>7.39%</td>
<td>6.45%</td>
</tr>
</tbody>
</table>

Summary (in seconds):

<table>
<thead>
<tr>
<th>Policy</th>
<th>Average Delay</th>
<th>Average Composite Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>10,833</td>
<td>17,869</td>
</tr>
<tr>
<td>Next-to-run</td>
<td>11,562</td>
<td>12,966</td>
</tr>
<tr>
<td>Preemption</td>
<td>10,350</td>
<td>11,019</td>
</tr>
</tbody>
</table>
CHAPTER 4: CLOUDS
## Virtual Machine Allocation

<table>
<thead>
<tr>
<th>Step</th>
<th>Eucalyptus Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Transfer</td>
<td>Images (including kernel and ramdisk, if applicable) may be cached on nodes. If so, image is copied from cache to location on disk where the instance will be hosted. If the image is not cached, it is downloaded from Walrus (image repository) via curl. After download, the image is copied to cache, if applicable.</td>
</tr>
<tr>
<td>Instance Preparation</td>
<td>This step includes adding the user’s ssh key to the image and creating the swap and ephemeral disks, which take up the remainder of the disk space available to the image that is not consumed by the kernel, ramdisk, and root image.</td>
</tr>
<tr>
<td>VM Startup</td>
<td>Eucalyptus interacts with KVM and Xen via the libvirt toolkit. The necessary XML file is created, and the instance is started via the appropriate libvirt function call.</td>
</tr>
</tbody>
</table>
Single Node VM Allocation

Average and standard deviations (in seconds) for measured allocation delay:

<table>
<thead>
<tr>
<th>VM Type</th>
<th>1 Instance</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
</tr>
<tr>
<td>m1.small</td>
<td>50</td>
<td>4.5</td>
<td>84</td>
<td>5.8</td>
<td>176</td>
<td>22.9</td>
<td>424</td>
</tr>
<tr>
<td>c1.medium</td>
<td>90</td>
<td>8.9</td>
<td>165</td>
<td>13.2</td>
<td>359</td>
<td>48.7</td>
<td>845</td>
</tr>
<tr>
<td>m1.large</td>
<td>152</td>
<td>16.8</td>
<td>299</td>
<td>29.9</td>
<td>637</td>
<td>87.5</td>
<td>-</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>274</td>
<td>34.7</td>
<td>568</td>
<td>71.6</td>
<td>1,325</td>
<td>191.5</td>
<td>-</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>276</td>
<td>34.6</td>
<td>573</td>
<td>67.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Breadboard Node Behavior

- **Purpose**: determine effect of Breadboard node on allocation delay
- **bb–prod cloud**
- **MG–RAST image**
  - c1.xlarge instance type
  - precached
- **Attempts to improve**:
  - Sparse file, XFS file system
Multiple Node VM Allocation

Average and standard deviations (in seconds) for measured allocation delay:

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>1 Instance per Node</th>
<th>2 Instances per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>1</td>
<td>429</td>
<td>8.2</td>
</tr>
<tr>
<td>2</td>
<td>447</td>
<td>41.2</td>
</tr>
<tr>
<td>4</td>
<td>443</td>
<td>30.1</td>
</tr>
<tr>
<td>8</td>
<td>492</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>616</td>
<td>26.1</td>
</tr>
</tbody>
</table>
Modified Multiple Node VM Allocation

- **Modifications:**
  - Sparse file for ephemeral disk
  - xfs file system over ext3
- **bb-prod cloud**
- **11 GB MG–RAST image**
  - c1.xlarge instance type
  - precached
Modified Multiple Node VM Allocation

Average and standard deviations (in seconds) for measured allocation delay:

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>1 Instance per Node</th>
<th>2 Instances per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>1</td>
<td>331</td>
<td>21.2</td>
</tr>
<tr>
<td>2</td>
<td>337</td>
<td>18.8</td>
</tr>
<tr>
<td>4</td>
<td>352</td>
<td>17.7</td>
</tr>
<tr>
<td>8</td>
<td>360</td>
<td>17.7</td>
</tr>
</tbody>
</table>
Magellan Default Priority Experiments

- **Magellan**
- **MGRAST image**
  - c1.xlarge type
    - Requires entire node
      - Disk: 100 GB vs. 20 GB
      - Mem: 16 GB vs. 2 GB
      - Cores: 8 vs. 4
  - precached
- **Quirks with Magellan:**
  - Random instance failures
  - Rest periods required

![Graph showing delay vs. number of instances](image)
Magellan Default
Priority Allocation

Average and standard deviations (in seconds) for measured allocation delay:

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>$\mu$</th>
<th>$\sigma$</th>
<th>Error</th>
<th>Rest Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>353</td>
<td>35.5</td>
<td>0%</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>355</td>
<td>52.4</td>
<td>0%</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>387</td>
<td>14.3</td>
<td>0%</td>
<td>20</td>
</tr>
<tr>
<td>32</td>
<td>444</td>
<td>50.5</td>
<td>0%</td>
<td>30</td>
</tr>
<tr>
<td>64</td>
<td>591</td>
<td>37.1</td>
<td>0%</td>
<td>75</td>
</tr>
<tr>
<td>128</td>
<td>831</td>
<td>67</td>
<td>5%</td>
<td>150</td>
</tr>
</tbody>
</table>
Magellan Default Priority with Eucalyptus 2.0

- Replication of previous experiment
  - MG–RAST, c1.xlarge, precached
- Much better stability and scalability
  - Very few random failures
  - Rest periods no longer needed
Average and standard deviations (in seconds) for measured allocation delay:

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>$\mu$</th>
<th>$\sigma$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>374</td>
<td>42.1</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>368</td>
<td>23.8</td>
<td>0%</td>
</tr>
<tr>
<td>16</td>
<td>385</td>
<td>22.2</td>
<td>0%</td>
</tr>
<tr>
<td>32</td>
<td>398</td>
<td>17.7</td>
<td>0%</td>
</tr>
<tr>
<td>64</td>
<td>406</td>
<td>15.2</td>
<td>0%</td>
</tr>
<tr>
<td>128</td>
<td>413</td>
<td>14.8</td>
<td>5%</td>
</tr>
</tbody>
</table>
Magellan Default Priority with OpenStack

- Previous experiment replicated
  - MG–RAST, c1.xlarge, precached
- Differences from Eucalyptus:
  - c1.xlarge
    - 60 GB more disk
    - 6 GB less mem
  - Sparse files
  - Copy-on-write format
Magellan Default Priority with OpenStack

Average and standard deviations (in seconds) for measured allocation delay:

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>$\mu$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>12</td>
<td>0.7</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>2.3</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>0.9</td>
</tr>
<tr>
<td>32</td>
<td>31</td>
<td>3.6</td>
</tr>
<tr>
<td>64</td>
<td>43</td>
<td>3.3</td>
</tr>
<tr>
<td>128</td>
<td>90</td>
<td>3.2</td>
</tr>
<tr>
<td>256</td>
<td>220</td>
<td>16</td>
</tr>
</tbody>
</table>
Preemption Breadboard
Single-Node Experiment

Allocation delays (in seconds):

<table>
<thead>
<tr>
<th>Preempted Instance Type</th>
<th>1 instance μ</th>
<th>1 instance σ</th>
<th>2 instances μ</th>
<th>2 instances σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROBE</td>
<td>253</td>
<td>22.5</td>
<td>435</td>
<td>13.9</td>
</tr>
<tr>
<td>m1.small</td>
<td>272</td>
<td>18.3</td>
<td>457</td>
<td>31.5</td>
</tr>
<tr>
<td>c1.medium</td>
<td>270</td>
<td>17.6</td>
<td>440</td>
<td>29.1</td>
</tr>
<tr>
<td>m1.large</td>
<td>264</td>
<td>19.4</td>
<td>450</td>
<td>22.6</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>272</td>
<td>25.5</td>
<td>451</td>
<td>21.3</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>264</td>
<td>22.2</td>
<td>454</td>
<td>42.3</td>
</tr>
</tbody>
</table>

Back
# Preemption Breadboard Single Node (from logs)

## Cost of preemption mechanism (in seconds):

<table>
<thead>
<tr>
<th>Preempted Instance Type</th>
<th>1 Instance</th>
<th>2 Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>2.32</td>
<td>4.28</td>
</tr>
<tr>
<td>c1.medium</td>
<td>2.72</td>
<td>4.84</td>
</tr>
<tr>
<td>m1.large</td>
<td>1.32</td>
<td>3.12</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>2.08</td>
<td>4.21</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>1.2</td>
<td>2.52</td>
</tr>
</tbody>
</table>

## Allocation delays (in seconds):

<table>
<thead>
<tr>
<th>Preempted Instance Type</th>
<th>1 Instance</th>
<th>2 Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>PROBE</td>
<td>253</td>
<td>22.5</td>
</tr>
<tr>
<td>m1.small</td>
<td>272</td>
<td>18.3</td>
</tr>
<tr>
<td>c1.medium</td>
<td>270</td>
<td>17.6</td>
</tr>
<tr>
<td>m1.large</td>
<td>264</td>
<td>19.4</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>272</td>
<td>25.5</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>264</td>
<td>22.2</td>
</tr>
</tbody>
</table>
# Preemption Breadboard: Multiple-Node Experiments

### Allocation delays (in seconds):

<table>
<thead>
<tr>
<th># of nodes</th>
<th>1 instance per node</th>
<th>2 instances per node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Probes</td>
<td>Urgent</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>1</td>
<td>264</td>
<td>23.5</td>
</tr>
<tr>
<td>4</td>
<td>318</td>
<td>18.2</td>
</tr>
<tr>
<td>8</td>
<td>328</td>
<td>11.4</td>
</tr>
<tr>
<td>16</td>
<td>342</td>
<td>17.1</td>
</tr>
</tbody>
</table>

![Graph showing allocation delays for different node configurations and allocation schemes](graph.png)
## Suspension Breadboard Single-Node Experiment

### Allocation delays (in seconds):

<table>
<thead>
<tr>
<th>Suspended Instance Type</th>
<th>1 instance</th>
<th>2 instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>PROBE</td>
<td>264</td>
<td>23.5</td>
</tr>
<tr>
<td>m1.small</td>
<td>291</td>
<td>20.3</td>
</tr>
<tr>
<td>c1.medium</td>
<td>305</td>
<td>17.5</td>
</tr>
<tr>
<td>m1.large</td>
<td>293</td>
<td>19</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>306</td>
<td>17.2</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>310</td>
<td>25.7</td>
</tr>
</tbody>
</table>

### Cost of suspension mechanism (in seconds, from log):

<table>
<thead>
<tr>
<th>Suspended Instance Type</th>
<th>1 Instance</th>
<th>2 Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>14.6</td>
<td>31.9</td>
</tr>
<tr>
<td>c1.medium</td>
<td>28</td>
<td>57.7</td>
</tr>
<tr>
<td>m1.large</td>
<td>21.5</td>
<td>36.8</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>30.8</td>
<td>65.8</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>33.1</td>
<td>55.1</td>
</tr>
</tbody>
</table>
Suspension Breadboard
Multiple-Node Experiments

Allocation delays (in seconds):

<table>
<thead>
<tr>
<th># of nodes</th>
<th>1 instance per node</th>
<th>2 instances per node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Probes</td>
<td>Urgent</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>1</td>
<td>264</td>
<td>23.5</td>
</tr>
<tr>
<td>4</td>
<td>318</td>
<td>18.2</td>
</tr>
<tr>
<td>8</td>
<td>328</td>
<td>11.4</td>
</tr>
<tr>
<td>16</td>
<td>342</td>
<td>17.1</td>
</tr>
</tbody>
</table>

Back
VM Migration

- Create space by migrating smaller, non-urgent VMs
- Xen supports live migration
  - Requires shared storage
- Files as virtual block devices
  - rsync, suspend, rsync, resume
- Benefit: shorter, less variable durations of unavailability
- Scheduler: limit one migration per source
### Migration Breadboard

#### Single-Migration Experiment

**Allocation delays (in seconds):**

<table>
<thead>
<tr>
<th>Migrated Instance Type</th>
<th>Allocation Delay</th>
<th>Unavailability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>PROBE</td>
<td>264</td>
<td>23.5</td>
</tr>
<tr>
<td>m1.small</td>
<td>350</td>
<td>17.9</td>
</tr>
<tr>
<td>c1.medium</td>
<td>430</td>
<td>22.3</td>
</tr>
<tr>
<td>m1.large</td>
<td>562</td>
<td>21.2</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>806</td>
<td>16</td>
</tr>
</tbody>
</table>

**Cost of suspension mechanism (in seconds, from log):**

<table>
<thead>
<tr>
<th>Migrated Instance Type</th>
<th>$\mu$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>80</td>
<td>2.9</td>
</tr>
<tr>
<td>c1.medium</td>
<td>155</td>
<td>2.4</td>
</tr>
<tr>
<td>m1.large</td>
<td>277</td>
<td>2.1</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>532</td>
<td>3</td>
</tr>
</tbody>
</table>
Migration Breadboard
Multiple-Migration Experiment

Allocation delay (in seconds):

<table>
<thead>
<tr>
<th># of nodes</th>
<th>Migrated instance type</th>
<th># migrations</th>
<th>Allocation delay</th>
<th>μ</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>m1.small</td>
<td>3</td>
<td>451</td>
<td>15.6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c1.medium</td>
<td>3</td>
<td>548</td>
<td>17.7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>m1.large</td>
<td>2</td>
<td>574</td>
<td>8.82</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>m1.xlarge</td>
<td>2</td>
<td>828</td>
<td>14.2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>m1.small</td>
<td>6</td>
<td>471</td>
<td>12.2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>c1.medium</td>
<td>6</td>
<td>568</td>
<td>14.4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>m1.large</td>
<td>4</td>
<td>610</td>
<td>14.2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>m1.xlarge</td>
<td>4</td>
<td>864</td>
<td>14.2</td>
<td></td>
</tr>
</tbody>
</table>
VM Quality of Service

- Dynamically reduce memory or virtual CPUs of running virtual machines
  - Memory and CPUs are more expensive and rarer
- Policy details: Release up to $\frac{1}{2}$ memory and/or $\frac{1}{2}$ number of CPUs allocated to targeted VMs
  - On Breadboard, CPUs always limiting attribute
VM QoS on NAS Benchmarks

- NAS benchmarks installed on CentOS c1.xlarge image (compiled for 4 CPUs)

### Allocation delays (in seconds)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Class</th>
<th>Normal Allocation</th>
<th>$\frac{1}{2}$ Cores</th>
<th>$\frac{1}{2}$ Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>bt</td>
<td>B</td>
<td>154</td>
<td>299</td>
<td>169</td>
</tr>
<tr>
<td>cg</td>
<td>C</td>
<td>161</td>
<td>313</td>
<td>176</td>
</tr>
<tr>
<td>ep</td>
<td>C</td>
<td>86</td>
<td>174</td>
<td>88</td>
</tr>
<tr>
<td>ft</td>
<td>B</td>
<td>40</td>
<td>69</td>
<td>fail</td>
</tr>
<tr>
<td>is</td>
<td>C</td>
<td>10</td>
<td>18</td>
<td>fail</td>
</tr>
<tr>
<td>lu</td>
<td>B</td>
<td>113</td>
<td>209</td>
<td>126</td>
</tr>
<tr>
<td>mg</td>
<td>B</td>
<td>7</td>
<td>23</td>
<td>9</td>
</tr>
<tr>
<td>sp</td>
<td>B</td>
<td>249</td>
<td>420</td>
<td>329</td>
</tr>
</tbody>
</table>
VM QoS Single-Node Experiments

Allocation delays and cost (from logs) in seconds:

<table>
<thead>
<tr>
<th>Targeted Instance Type</th>
<th>Allocation Delay</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROBE</td>
<td>264</td>
<td>-</td>
</tr>
<tr>
<td>m1.large</td>
<td>278</td>
<td>0.52</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>274</td>
<td>0.44</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>278</td>
<td>0.24</td>
</tr>
</tbody>
</table>

Back
## VM QoS Multiple-Node Experiments

### Allocation delays (in seconds):

<table>
<thead>
<tr>
<th># of nodes</th>
<th>Probes $\mu$</th>
<th>Probes $\sigma$</th>
<th>Urgent instances $\mu$</th>
<th>Urgent instances $\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>264</td>
<td>23.5</td>
<td>278</td>
<td>19.7</td>
</tr>
<tr>
<td>4</td>
<td>318</td>
<td>18.2</td>
<td>322</td>
<td>14.8</td>
</tr>
<tr>
<td>8</td>
<td>328</td>
<td>11.4</td>
<td>336</td>
<td>10.3</td>
</tr>
<tr>
<td>16</td>
<td>342</td>
<td>17.1</td>
<td>354</td>
<td>10.3</td>
</tr>
</tbody>
</table>

![Graph showing average delay with number of nodes](image)

**Back**
Master Index

• Introduction
  ◦ Definition and Motivation
  ◦ Problem Statement
  ◦ Related Work

• SPRUCE: Urgent Computing Framework

• Resource Selection

• Clouds as Urgent Computing Resources

• Contributions, Conclusions, and Future Work