Multi-State Grid Resource Availability Characterization

Brent Rood and Michael J. Lewis

Grid Computing Research Lab
Department of Computer Science
Binghamton University (SUNY)
Resource Heterogeneity

- **Computational power**
  - CPU clock rate and type, memory, disk space, etc.
  - Local load

- **Availability**
  - Dedicated and highly available
  - Non-dedicated and intermittent

- **Policy and Autonomy**
  - When and how the resource is available to the grid is dictated by the owner
Application Heterogeneity

- **Duration**
  - Applications vary in execution time

- **Checkpointability**
  - Applications may or may not be able to take a checkpoint
The Problem

- Tough to get the right match between resources and applications when scheduling
  - Heterogeneous resources
  - Application characteristics differ
  - Unreliable resource availability
  - Resources can become unavailable in various ways
Approach

- *Trace analysis method* exposes more details about the way a resource fails
- *Sample* Condor trace
- Resource availability *classification* approach
- *Class-based* scheduling results
  - Results show 31% improvement in makespan by considering just resource availability
- An initial simple *availability predictor* implementation
Related Work - Traces

- Workstation Pool and Grid Environment Traces
  - Examines resource characteristics
  - Analyze availability and unavailability
  - E.g. Microsoft, PlanetLab, BOINC, etc.

- However
  - Machine availability is viewed as either available or unavailable
  - This doesn’t consider the various causes of unavailability
Related Work - Prediction

- **Availability Prediction**
  - Predict when a machine will become unavailable

- **Problem**
  - Previous prediction methods predict just the availability of a resource and **not the way** in which the resource is likely to become unavailable.

- **Joanne Ren developed a multi-state model for resource availability - closest related**
  - Model uses 3 states for various levels of local CPU load, 1 memory thrashing state and 1 unavailability state
  - Our model is also multi-state but involves different states as well as contrasts in how those states are utilized
Multi-state model motivation: Condor

- **Condor**
  - Software infrastructure: Harness idle resources with facilities for automatic checkpointing and migration

- **Condor Defaults (simplified)**
  - Application begins execution on a resource
    - Idle from user activity for at least 15 minutes
    - Local CPU load < 30%
  - Application suspension
    - User activity or CPU load > 50%
  - Resume suspended application
    - Resource idle for 5 minutes and local CPU load < 30%
  - Job eviction (removed from resource; checkpoint taken if possible)
    - Suspension for > 10 minutes or resource shutdown

- Not all jobs can be checkpointed
### Condor Application Execution

<table>
<thead>
<tr>
<th>Condition</th>
<th>Checkpointable</th>
<th>Non-Checkpointable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Suspension</strong></td>
<td>Halts Execution</td>
<td>Halts Execution</td>
</tr>
<tr>
<td>(User presence or CPU load)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Job Removal-Graceful Eviction</strong></td>
<td>Checkpoint taken and application migrated to new resource</td>
<td>Restart from beginning on new resource</td>
</tr>
<tr>
<td>(Suspended too long or resource shutdown)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Machine Failure-Ungraceful Eviction</strong></td>
<td>Most recent periodic checkpoint migrated to new resource</td>
<td>Restart from beginning on new resource</td>
</tr>
</tbody>
</table>
Multi-State Availability Model

Available to Grid → User Present
User Present → Job Eviction or Graceful Shutdown
Job Eviction or Graceful Shutdown → Unavailable
Unavailable → CPU Threshold Exceeded
CPU Threshold Exceeded → Available to Grid
Investigation Methodology

- Analyze traces to expose trends
- Classify resources by how they behave
- Schedule based on the resource classes
- Develop and evaluate an initial multi-state availability predictor

Goal
  - Show that both when a resource is available and *how* it becomes unavailable can be exploited for performance gain
Trace

- Obtained four months of data for the University of Notre Dame’s Condor pool in early 2007
- Time-stamped CPU load and keyboard idle time measurements every 16 minutes.
- **Goal**: Identify trends that characterize the behavior of resources in the multi-state model.
Condor Pool Characteristics

- **Availability**
- **User Presence**
- **CPU Threshold**
- **Unavailable**
Condor Pool Characteristics Cont’d

Percentage of time spent in each state (total):
- Available (84%)
- CPU Threshold Exceeded (3%)
- User Present (3%)
- Unavailable (9%)
• Transitions dictate when an executing application may need to find another resource
  – The type of transition indicates if a checkpoint can be taken (Ungraceful vs. Graceful eviction)
• We report the number of transitions for each state as a function of
  – Day of the week
  – Hour of the day
Daily Transitional Behavior

Availability

User Presence

CPU Threshold

Unavailable
Hourly Transitional Behavior

**Availability**

- Number of Transitions to Available
- Hour of Day

**User Presence**

- Number of Transitions to User Present
- Hour of Day

**CPU Threshold**

- Number of Transitions to CPU Threshold Exceeded
- Hour of Day

**Unavailable**

- Number of Transitions to Unavailable
- Hour of Day
Machine Classification

- **Goal:** Classify resources by considering multiple availability characteristics simultaneously
- **Approach:** Organize resources based on which type of applications they would be most useful executing
  - Average availability duration
  - How a machine transitions to unavailable (gracefully or ungracefully)
## Machine Classification Cont’d

<table>
<thead>
<tr>
<th>Average Availability Duration</th>
<th>Average # of Graceful Transitions</th>
<th>Ungraceful Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>High</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Medium High</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Medium Low</td>
<td>22</td>
<td>18</td>
</tr>
<tr>
<td>Low</td>
<td>98</td>
<td>32</td>
</tr>
</tbody>
</table>
Classification Implications

- Simulated 100 job runtimes equally distributed between 5 minutes to 12 hours
- Execute each job runtime on each machine 1000 times, beginning at a random point in the machine’s trace for each of the four months
- Intuition:
  - The higher availability classes are more suited to longer running applications
Classification Implications Cont’d

Graceful Failure Rate vs. Job Duration

- HA-LG-LUG
- MHA-LG-LUG
- MLA-LG-HUG
- LA-HG-HUG

Failure Rate (%) vs. Job Duration (Instructions)

2 x 10^7
Class-aware Scheduling

- A scheduler should match:
  - Application duration with the expected availability duration of a resource
  - Application checkpointability with a resource’s most likely way of transitioning to unavailable
    - Checkpointable applications can tolerate graceful transitions because they can checkpoint.
    - Non-checkpointable applications cannot tolerate graceful or ungraceful transitions
Class-aware Scheduling- Simulation

- 2000 jobs of a random length are created (between 5 minutes and 12 hours) and inserted into system at random times during months 3 and 4
- 50% checkpointable; 50% non-checkpointable
- Two schedulers that mapped the exact same random job set onto the resources
  - Random scheduler maps jobs onto random available resources (Condor default)
  - Class-based scheduler uses classifications of machines to decide which machine to map the job onto
    - Classifies machines in months 1 and 2
- Schedulers chosen to highlight performance implications of just considering availability
# Class-aware Scheduling - Results

<table>
<thead>
<tr>
<th></th>
<th>Random Scheduler</th>
<th>Classifier Scheduler</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completed Jobs</td>
<td>1883.2</td>
<td>1936.0</td>
<td>2.27%</td>
</tr>
<tr>
<td>Gracefully Evicted Jobs</td>
<td>1244.0</td>
<td>492.0</td>
<td>60.45%</td>
</tr>
<tr>
<td>Ungracefully Evicted Jobs</td>
<td>28.6</td>
<td>14.3</td>
<td>50.0%</td>
</tr>
<tr>
<td>Avg. Execution time (secs.)</td>
<td>71209.0</td>
<td>48656.0</td>
<td>31.6%</td>
</tr>
</tbody>
</table>
Prediction

- The previous results depend on knowing the machine classes…
- Classification scheme shortcoming
  - Doesn’t account for the patterns/trends of state behavior for each machine
- Solution:
  - Instead of classification, make predictions about a resource’s future availability
Multi-state Availability Prediction

- Given a resource and an interval, predict if the resource will complete the interval or in what state it will exit availability
- “Yesterday predictor”
  - Previous weekday/weekend behavior analysis
- **Goal:** Show even a simple predictor can obtain decent accuracy
  - Worth noting: subsequent work has surpasses these accuracies
- Simulated making 3 million predictions at random times on random resources
- Results report the accuracy of each prediction type given the duration of the prediction interval
Initial Prediction Results
Conclusions

- The way in which a resource transitions to unavailable effects application performance.
- Different categories of machines exhibit different availability characteristics.
- Class-aware schedulers can achieve performance gain.
- The classes and states seem somewhat predictable.