Falkon: a Fast and Light-weight task execution framework for Grid Environments

Ioan Raicu
Distributed Systems Laboratory
Computer Science Department
University of Chicago

Joint work with:
Yong Zhao: Univ. of Chicago, CS
Catalin Dumitrescu: Univ. of Chicago, CS
Ian Foster: Univ. of Chicago, CS & CI, Argonne National Laboratory, MCS
Mike Wilde: University of Chicago, CI, Argonne National Laboratory, MCS

Funded by:
NASA Ames Research Center GSRP: October 2006 – September 2007

CS Seminar
University of Chicago, Department of Computer Science
April 30th, 2007
Grid Computing

• Grid Computing’s focus:
  – large-scale resource sharing: direct access to computers, software, data
  – innovative applications
  – high-performance orientation

• The ‘Grid problem’:
  – Definition: flexible, secure, and coordinated resource sharing among dynamic collections of individuals, institutions, and resources
  – Challenges: Security (Authentication, Authorization), resource management (resource access, resource discovery, scheduling, data management)
Falkon: a Fast and Light-weight task execution framework

- **Goal**: enable the rapid and efficient execution of many independent jobs on large compute clusters

- Combines two techniques:
  - *multi-level scheduling* techniques to enable separate treatments of resource provisioning
  - a *streamlined task dispatcher* able to achieve order-of-magnitude higher task dispatch rates than conventional schedulers
Execution Model

• Dispatch policy
  – Round-robin

• Replay policy

• Resource Acquisition Policy
  – One-at-a-time
  – Additive
  – Exponential
  – All-at-once
  – Optimal

• Resource Release Policy
  – Centralized
  – Distributed
Falkon 2-Tier Architecture

- Tier 1: Dispatcher
  - GT4 Web Service accepting task submissions from clients and sending them to available executors

- Tier 2: Executor
  - Run tasks on local resources

- Provisioner
  - Static and dynamic resource provisioning
Falkon Message Exchanges

- **Description:**
  1. task(s) submit
  2. notification for work
  3. pick up task(s)
  4. deliver task(s) results
  5. notification for task result
  6. pick up task(s) results

- **Worst case:**
  - 4 WS messages (1, 3, 4, 6) and 2 notifications (2, 5) per task

- **Best case:**
  - 1 WS message (4) and 1 notification (5) message per task
Enhancements

• Bundling
  – Include multiple tasks per communication message

• Piggy-Backing
  – Attach next task to acknowledgement of previous task

• Message reduction:
  – Now: 6 → 2
  – General Lower Bound: 6 → 1
  – Application Specific Lower Bound: 6 → c, where c is a small positive value close to 0
### Testbeds

<table>
<thead>
<tr>
<th>Name</th>
<th># of Nodes</th>
<th>Processors</th>
<th>Memory</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>TG_ANL_IA32_CLUSTER</td>
<td>98</td>
<td>Dual Xeon 2.4GHz</td>
<td>4GB</td>
<td>1Gb/s</td>
</tr>
<tr>
<td>TG_ANL_IA64_CLUSTER</td>
<td>64</td>
<td>Dual Itanium 1.5GHz</td>
<td>4GB</td>
<td>1Gb/s</td>
</tr>
<tr>
<td>TP_UC_x64_Cluster</td>
<td>122</td>
<td>Dual Opteron 2.2GHz</td>
<td>4GB</td>
<td>1Gb/s</td>
</tr>
<tr>
<td>UC_x64</td>
<td>1</td>
<td>Dual Xeon 3GHz w/ HT</td>
<td>2GB</td>
<td>100 Mb/s</td>
</tr>
<tr>
<td>UC_IA32</td>
<td>1</td>
<td>Intel P4 2.4GHz</td>
<td>1GB</td>
<td>100 Mb/s</td>
</tr>
</tbody>
</table>
Performance Metrics

- Throughput (tasks/sec)
- Time to execute per task (ms)
- Execution time for a given set of tasks
- Speedup
- Efficiency
Falkon Scalability

- 2M tasks processed (sleep 0) on 16 machines (32 executors)
- 7490 seconds ~ 267 tasks/sec
- Queue Length ~ 1.5M
- Sun JDK 1.4.2 with 1.5GB Heap
- Throughput variations → issues with GC
Falkon Scalability

- 54K tasks processed (sleep 480) on 60 machines (54K executors)
- 888 seconds ~ 61 tasks/sec
- Per task overhead: 127 ms, standard deviation 76ms
- Sun JDK 1.4.2 with 1.5GB Heap
Security Overhead

- **Experiment:**
  - 1 client, 1 dispatcher, 1 executor
  - 30 tasks of sleep 60
- **Security:**
  - None
  - GSI Transport with authentication and encryption
  - GSI Secure Conversation with authentication and encryption

- **MyCluster Comparison**
  - Condor: 5%
  - SGE: 25%

<table>
<thead>
<tr>
<th></th>
<th>Exec Time (sec)</th>
<th>Exec Overhead %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal Tasks Execution</td>
<td>1800.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>No Security</td>
<td>1803.46</td>
<td>0.19%</td>
</tr>
<tr>
<td>GSI Transport (Auth + Enc)</td>
<td>1817.37</td>
<td>0.96%</td>
</tr>
<tr>
<td>GSI Secure Conversation (Auth + Enc)</td>
<td>1815.58</td>
<td>0.87%</td>
</tr>
</tbody>
</table>
Dynamic Resource Provisioning

- Synthetic Workload
  - 18 stages
  - 1000 tasks
  - 17,820 CPU seconds
  - 1,260 seconds total time on 32 machines

<table>
<thead>
<tr>
<th>Stage #</th>
<th># of Tasks</th>
<th>Task Time (sec)</th>
<th>Stage Time (32 machines)</th>
<th># Machines (32 max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>60</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>60</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>60</td>
<td>60</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>60</td>
<td>60</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>60</td>
<td>60</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>60</td>
<td>60</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>64</td>
<td>60</td>
<td>120</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>120</td>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>640</td>
<td>6</td>
<td>120</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>160</td>
<td>12</td>
<td>60</td>
<td>32</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>60</td>
<td>60</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>60</td>
<td>60</td>
<td>20</td>
</tr>
<tr>
<td>13</td>
<td>18</td>
<td>60</td>
<td>60</td>
<td>18</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>60</td>
<td>60</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>60</td>
<td>60</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>60</td>
<td>60</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>60</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>60</td>
<td>60</td>
<td>1</td>
</tr>
</tbody>
</table>
Dynamic Resource Provisioning

- Compared:
  - GRAM+PBS: 1 job per task
  - Falkon-15, -60, -120, -180: dynamic resource provisioning – 15, 60, 120, and 180 seconds idle time before resource de-allocation, and with 32 maximum machines
  - Falkon-\(\infty\): static resource provisioning with 32 maximum machines
  - Ideal with 32 maximum machines

- Falkon-15 / Falkon-180:
  - Allocated (blue): LRM wait queue
  - Registered (red): idle executors
  - Active (green): executors processing tasks
Dynamic Resource Provisioning

- **End-to-end execution time:**
  - 1260 sec in ideal case
  - 4904 sec $\rightarrow$ 1276 sec

- **Average task queue time:**
  - 42.2 sec in ideal case
  - 611 sec $\rightarrow$ 43.5 sec

- **Trade-off:**
  - Resource Utilization for Execution Efficiency

<table>
<thead>
<tr>
<th></th>
<th>GRAM +PBS</th>
<th>Falkon-15</th>
<th>Falkon-60</th>
<th>Falkon-120</th>
<th>Falkon-180</th>
<th>Falkon-$\infty$</th>
<th>Ideal (32 nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Queue Time (sec)</strong></td>
<td>611.1</td>
<td>87.3</td>
<td>83.9</td>
<td>74.7</td>
<td>44.4</td>
<td>43.5</td>
<td>42.2</td>
</tr>
<tr>
<td><strong>Execution Time (sec)</strong></td>
<td>56.5</td>
<td>17.9</td>
<td>17.9</td>
<td>17.9</td>
<td>17.9</td>
<td>17.9</td>
<td>17.8</td>
</tr>
<tr>
<td><strong>Execution Time %</strong></td>
<td>8.5%</td>
<td>17.0%</td>
<td>17.6%</td>
<td>19.3%</td>
<td>28.7%</td>
<td>29.2%</td>
<td>29.7%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>GRAM +PBS</th>
<th>Falkon-15</th>
<th>Falkon-60</th>
<th>Falkon-120</th>
<th>Falkon-180</th>
<th>Falkon-$\infty$</th>
<th>Ideal (32 nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time to complete (sec)</strong></td>
<td>4904</td>
<td>1754</td>
<td>1680</td>
<td>1507</td>
<td>1484</td>
<td>1276</td>
<td>1260</td>
</tr>
<tr>
<td><strong>Resource Utilization</strong></td>
<td>30%</td>
<td>89%</td>
<td>75%</td>
<td>65%</td>
<td>59%</td>
<td>44%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Execution Efficiency</strong></td>
<td>26%</td>
<td>72%</td>
<td>75%</td>
<td>84%</td>
<td>85%</td>
<td>99%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Resource Allocations</strong></td>
<td>1000</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>#Jobs/workflow</th>
<th>#Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATLAS: High Energy Physics Event Simulation</td>
<td>500K</td>
<td>1</td>
</tr>
<tr>
<td>fMRI DBIC: AIRSN Image Processing</td>
<td>100s</td>
<td>12</td>
</tr>
<tr>
<td>FOAM: Ocean/Atmosphere Model</td>
<td>2000</td>
<td>3</td>
</tr>
<tr>
<td>GADU: Genomics</td>
<td>40K</td>
<td>4</td>
</tr>
<tr>
<td>HNL: fMRI Aphasia Study</td>
<td>500</td>
<td>4</td>
</tr>
<tr>
<td>NVO/NASA: Photorealistic Montage/Morphology</td>
<td>1000s</td>
<td>16</td>
</tr>
<tr>
<td>QuarkNet/I2U2: Physics Science Education</td>
<td>10s</td>
<td>3 ~ 6</td>
</tr>
<tr>
<td>RadCAD: Radiology Classifier Training</td>
<td>1000s</td>
<td>5</td>
</tr>
<tr>
<td>SIDGrid: EEG Wavelet Processing, Gaze Analysis</td>
<td>100s</td>
<td>20</td>
</tr>
<tr>
<td>SDSS: Coadd, Cluster Search</td>
<td>40K, 500K</td>
<td>2, 8</td>
</tr>
</tbody>
</table>

• Applications tested:
  – Medicine: fMRI
  – Astronomy: Montage

• Compared:
  – GRAM: Swift submitting each task as a GRAM job
  – GRAM/Clustering: Swift submitting clusters of tasks as GRAM jobs
  – MPI: Executing the entire application as an MPI program
  – Falkon: Swift submitting each task as a task to Falkon
Applications: fMRI

- GRAM vs. Falkon: 85%~90% lower run time
- GRAM/Clustering vs. Falkon: 40%~74% lower run time
Applications: Montage

- GRAM/Clustering vs. Falkon: 57% lower application run time
- MPI* vs. Falkon: 4% higher application run time
- * MPI should be lower bound
Future Work

• Task pre-fetching
  – Overlap communication with computation

• Languages and Technologies
  – Move from Java to C
  – Move from Web Services to proprietary TCP-based protocol on the internals of Falkon

• Data Management
  – Data caching at executor local disk
    • Caching strategies: LRU, FIFO, popularity, …
  – Data replication at Grid site shared file systems
    • Replication strategies to meet a desired QoS
Future Work

• 3-Tier Architecture
  – Overview
    • Tier 1: Forwarder
      – Accept task submission from client(s) and forward them to appropriate dispatcher(s)
    • Tier 2: Dispatcher
      – Same as current Tier 1, but every Grid site (cluster) would have a separate dispatcher
    • Tier 3: Executor
      – Same as current Tier 2, and every executor would only communicate with respective local site dispatcher
  – Increases performance and scalability
  – Ensures that Falkon works with local access policies (firewalls, private IP spaces, etc)
More Information

- **Collaborators:**
  - Ioan Raicu, Computer Science Dept., The University of Chicago
  - Yong Zhao, Computer Science Dept., The University of Chicago
  - Catalin Dumitrescu, Computer Science Dept., The University of Chicago
  - Ian Foster, Math and Computer Science Div., Argonne National Laboratory & Computer Science Dept., The University of Chicago
  - Mike Wilde, Computation Institute, University of Chicago & Argonne National Laboratory

- **Contact information:**
  - Falkon specific: iraicu@cs.uchicago.edu
  - Swift Specific: yongzh@cs.uchicago.edu

- **Web:** [http://people.cs.uchicago.edu/~iraicu/research/Falkon/](http://people.cs.uchicago.edu/~iraicu/research/Falkon/)
- **Source Code:** [http://people.cs.uchicago.edu/~iraicu/research/Falkon/Falkon_v0.8.tgz](http://people.cs.uchicago.edu/~iraicu/research/Falkon/Falkon_v0.8.tgz)
- **Related Projects:**
  - Swift: [http://www.ci.uchicago.edu/swift/index.php](http://www.ci.uchicago.edu/swift/index.php)

- **Documents:**
  - Yong Zhao, Mihael Hategan, Ioan Raicu, Mike Wilde, Ian Foster. “Swift: a Parallel Programming Tool for Large Scale Scientific Computations”, under review at Scientific Programming Journal, Special Issue on Dynamic Computational Workflows: Discovery, Optimization, and Scheduling.