ComplexHPC Spring School
Day 2: KOALA Tutorial

The KOALA Scheduler

Nezih Yigitbasi
Delft University of Technology

May 10, 2011
Outline

• Koala Architecture
  • Job Model
  • System Components

• Support for different application types
  • Parallel Applications
  • Parameter Sweep Applications (PSAs)
  • Workflows
Introduction

- Developed in the DAS system
- Has been deployed on the DAS-2 in September 2005
- Ported to DAS-3 in April’07, and to DAS-4 in April’11
- Independent from grid middlewares such as Globus
- Runs on top of local schedulers

**Objectives:**
- Data and processor co-allocation in grids
- Supporting different application types
- Specialized job placement policies
Background (1): DAS-4

Operational since oct. 2010

- 1,600 cores (quad cores)
- 2.4 GHz CPUs
- accelerators
- 180 TB storage
- Infiniband
- Gb Eternet
Background (2): Grid Applications

- Different **application types** with different characteristics:
  - Parallel applications
  - Parameter sweep applications
  - Workflows
  - Data intensive applications

- **Challenges:**
  - Application characteristics and needs
  - Grid infrastructure is highly heterogeneous
  - Grid infrastructure configuration issues
  - Grid resources are highly dynamic
Koala Job Model

- A job consists of one or more job components

- A job component contains:
  - An executable name
  - Sufficient information necessary for scheduling
  - Sufficient information necessary for execution

<table>
<thead>
<tr>
<th>Fixed job</th>
<th>Non-fixed job</th>
</tr>
</thead>
<tbody>
<tr>
<td>job components</td>
<td></td>
</tr>
<tr>
<td>job component placement fixed</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flexible job</th>
</tr>
</thead>
<tbody>
<tr>
<td>same total job size</td>
</tr>
<tr>
<td>scheduler decides on split up and placement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fixed job</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduler decides on component placement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flexible job</th>
</tr>
</thead>
<tbody>
<tr>
<td>same total job size</td>
</tr>
<tr>
<td>scheduler decides on split up and placement</td>
</tr>
</tbody>
</table>
Koala Architecture (1)
Koala Architecture (2):
A Closer Look

- PIP/NIP: information services
- RLS: replica location service
- CO: co-allocator
- PC: processor claimer

- RM: run monitor
- RL: runners listener
- DM: data mover
- Ri: runners

A Closer Look
Scheduler

- Enforces **Scheduling Policies**
  - Co-Allocation Policies
    - Worst-Fit, Flexible Cluster Min., Comm. Aware, Close to Files
  - Malleability Management Policies
    - Favour Previously Started Malleable Applications
    - Equi Grow Shrink
  - Cycle Scavenging Policies
    - Equi-All, Equi-PerSite
  - Workflow Scheduling Policies
    - Single-Site, Multi-Site
Runners

- Extends support for different application types

- **KRunner**: Globus runner
- **PRunner**: A simplified job runner
- **IRunner**: Ibis applications
- **OMRunner**: OpenMPI applications
- **MRunner**: Malleable applications based on the DYNACO framework
- **WRunner**: For workflows (Directed Acyclic Graphs) and BoTs
The Runners Framework
Support for Different Application Types

- Parallel Applications
  - MPI, Ibis,...
  - Co-Allocation
  - Malleability

- Parameter Sweep Applications
  - Cycle Scavenging
  - Run as low-priority jobs

- Workflows
Support for Co-Allocation

- What is co-allocation (just to remind)
- Co-allocation Policies
- Experimental Results
Co-Allocation

• Simultaneous allocation of resources in multiple sites
  • Higher system utilizations
  • Lower queue wait times

• Co-allocated applications might be less efficient due to the relatively slow wide-area communications
  • Parallel applications may have different communication characteristics
Co-Allocation Policies (1)

• Dictate where the components of a job go

• Policies for **non-fixed jobs**:
  
  • **Load-aware**:
    (balance load in clusters)
  
  • **Input-file-location-aware**:
    (reduce file-transfer times)
  
  • **Communication-aware**:
    (reduce number of wide-area messages)

Co-Allocation Policies (2)

• Placement policies for **flexible jobs**:
  
  • **Queue time-aware**: Flexible Cluster Minimization
    (CM + reduce queue wait time)
    (FCM)

  • **Communication-aware**: Communication Aware (CA)
    (decisions based on inter-cluster communication speeds)

Co-Allocation Policies (3)

WF

Components

C1 (16)  C2 (16)  C3 (16)

Clusters

I  II  III

FCM

Component

C1 (16)  C2 (16)  C3 (16)
Experimental Results: Co-Allocation Vs. No co-allocation

- OpenMPI + DRMAA
- no co-allocation (left) vs. co-allocation (right)
- workloads of real parallel applications range from computation-intensive (Prime) to very communication-intensive (Wave)

### Average Job Response Time (s)

<table>
<thead>
<tr>
<th></th>
<th>Prime</th>
<th>Poisson</th>
<th>Wave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait</td>
<td>100</td>
<td>120</td>
<td>150</td>
</tr>
<tr>
<td>Execution</td>
<td>150</td>
<td>120</td>
<td>250</td>
</tr>
</tbody>
</table>

Co-allocation is disadvantageous for communication-intensive applications.
Experimental Results: The performance of the policies

- Flexible Cluster Min. vs. Comm. Aware
- Workloads of communication-intensive applications

considering the network metrics improves the co-allocation performance
Support for PSAs in Koala

- Background
- System Design
- Scheduling Policies
- Experimental Results
Parameter Sweep Application Model

- A single executable that runs for a large set of parameters
  - E.g.; monte-carlo simulations, bioinformatics applications...

- PSAs may run in multiple clusters simultaneously

- We support OGF’s JSDL 1.0 (XML)
Motivation

• How to run **thousands of tasks** in the DAS?

• Issues:
  • 15 min. rule!
  • Observational scheduling
  • Overload

• Run them as **Cycle Scavenging Applications !!**
  • Sets priority classes implicitly
  • No worries for observing empty clusters
Cycle Scavenging

- The technology behind **volunteer computing** projects
- Harnessing idle CPU cycles from desktops
- Download a software (screen saver)
  - Receive tasks from a central server
  - Execute a task when the computer is idle
  - Immediate preemption when the user is active again

[Logos of SETI@HOME, Compute Against Cancer, Fight AIDS @home, and TUDelft]
System Requirements

1. Unobtrusiveness
Minimal delay for (higher priority) local and grid jobs

2. Fairness
Multiple cycle scavenging applications running concurrently should be assigned comparable CPU-Time

3. Dynamic Resource Allocation
Cycle scavenging applications has to Grow/Shrink at runtime

4. Efficiency
As much use of dynamic resources as possible

5. Robustness and Fault Tolerance
Long-running, complex system: problems will occur, and must be dealt with
System Interaction

CS Policies:
- Equi-All: grid-wide basis
- Equi-PerSite: per cluster

Application Level Scheduling:
- Pull-based approach
- Shrinkage policy
Cycle Scavenging Policies

1. Equipartition-All
2. Equipartition-PerSite

Clusters

C1 (12)  C2 (12)  C3 (24)
Experimental Results

- DAS3
- Using Launchers vs. not
- 60s. dummy tasks
- Tested on a 32-node cluster

- Equi-All vs. Equi-PerSite
- 3 CS Users submit the same application with the same parameter range
- Non-CS Workloads: WBlock, WBurst

Support for Workflows in Koala

- Applications with dependencies
  - e.g., Montage workflow
    - Astronomy application to generate mosaics of the sky
    - 4500 tasks

- Dependencies are file transfers

- Experience the WRunner in the **hands-on** session
1. **Round Robin**: submits the *eligible* tasks to the clusters in round-robin order

2. **Single Cluster**: maps every complete workflow to the least-loaded cluster at its submission

3. **All Clusters**: submits each eligible task to the least loaded cluster
4. **All Clusters File-Aware**: submits each eligible task to the cluster that minimizes the transfer costs of the files on which it depends.

5. **Coarsening**: iteratively reduces the size of a graph by collapsing groups of nodes and their internal edges.
   - We use Heavy Edge Matching technique to group tasks that are connected with heavy edges.

---

6. **Cluster Minimization**: submits as many eligible tasks as possible to a cluster before considering the next cluster.

7. **HEFT* (Heterogeneous Earliest-Finish-Time)**: selects the task with the highest **upward rank** value and assigns the selected task to the cluster that ensures its earliest completion time.

**We will use Single Cluster and All Clusters in the hands-on session**

---

Workflow Engine Architecture

Workflow Description

Execution

SSH + Custom Protocol

DRMAA

DFS
Cloud Integration

- Resource management issues
  - When and how many resources to allocate
  - What type of resource to allocate
  - Which availability zone

- Performance issues
  - No queueing but resource acquisition/release overheads
  - Virtualization overhead
  - Wide-area file transfers

- Cost issues (for public clouds)
- Functionality Issues (MapReduce)
Conclusion

- **Koala** supports multiple application types:
  - Sequential applications
  - Parallel applications that may need co-allocation
  - Parallel applications that can grow/shrink at runtime
  - Parameter sweep applications
  - Workflows

- Different scheduling policies for each application type
  - No one size fits all policy
Questions? Comments?

“M.N.Yigitbasi@tudelft.nl”
http://www.st.ewi.tudelft.nl/~nezih/

More Information:

- Koala Project: http://st.ewi.tudelft.nl/koala
- PDS publication database: http://www.pds.twi.tudelft.nl
- DAS-4: http://www.cs.vu.nl/das4/