The KOALA Grid Scheduler

Ozan Sonmez

Parallel and Distributed Systems Group (PDS)
Department of Software Technology
Faculty EEMCS
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 2007
- 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):
DAS3

- VU (85 nodes)
- UvA/MultimediaN (46)
- UvA/VL-e (41)
- TU Delft (68)
- Leiden (32)

SURFnet6

10 Gb/s lambdas

- 272 AMD Opteron nodes
  792 cores, 1TB memory
- Heterogeneous:
  2.2-2.6 GHz
  single/dual core nodes
- Myrinet-10G (excl. Delft)
- Gigabit Ethernet
Background (2):
Grid Applications

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

- **Challenges:**
  - Applications have special 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

### Fixed Job
- Job components
- Job component placement fixed

### Non-fixed job
- Scheduler decides on component placement

### Flexible job
- Same total job size
- Scheduler decides on split up and placement

13-5-2009
Koala Architecture (1)
Koala Architecture (2):
Job flow with four phases

Phase 1: job placement
Phase 2: file transfer
Phase 3: claim processors
Phase 4: launch job
Koala Architecture (3):
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

13-5-2009
Scheduler

- Enforces **Scheduling Policies**
  - Co-Allocation Policies
    - WF, FCM, CA, CF
  - Malleability Management Policies
    - FPSMA, EGS
  - Cycle Scavenging Policies
    - Equi-All, Equi-PerSite
  - Workflow Management Policies
    - Single-Site, Multiple-Site
Runners

• Extends support for different job types

• **KRunner**: Globus runner
• **PRunner**: A simplified job runner
• **IRunner**: Ibis applications
• **OMRunner**: OpenMPI applications
• **MRunner**: Malleable applications (DYNACO framework)
• **WRunner**: For workflows (Directed Acyclic Graphs)
The Runners Framework

- Koala Scheduler
  - AbstractRunners
    - DRMAARunners
      - R1
      - R2
      - Rn
    - GlobusRunners
      - R1
      - R2
      - Rn
    - GATRunners
      - R1
      - R2
      - Rn
    - ??Runners
      - R1
      - R2
      - Rn
Writing your own Runner

```java
package org.koala.runnersFramework.runners;

import org.koala.internals.JCompRunnerInfo;
import org.koala.runnersFramework.DRMAARunners;

public class TestRunner extends DRMAARunners {

    public TestRunner() {
    }

    @Override
    public void parseParameters(String[] args) {
    }

    @Override
    public boolean prePhase() {
        return false;
    }
}
```
Writing your own Runner

```java
@Override
public void precondSubmit(JCompRunnerInfo component) {
}

@Override
public int submitComponent(JCompRunnerInfo component) {
    return 0;
}

@Override
public void postPhase(boolean jobRunWasSuccessful) {
}

/**
 * @param args
 */
public static void main(String[] args) {
}
```
Support for Different Application Types

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

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

- Workflows (new!)
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**: Worst Fit (WF)  
    (balance load in clusters)
  • **Input-file-location-aware**: Close-to-Files (CF)  
    (reduce file-transfer times)
  • **Communication-aware**: Cluster Minimization (CM)  
    (reduce number of wide-area messages)

Co-Allocation Policies (2)

• Placement policies for **flexible jobs**:
  
  • **Queue time-aware:** Flexible Cluster
    
    (CM + reduce queue wait time)
    
    Minimization (**FCM**)  
  
  • **Communication-aware:** Communication
    
    (decisions based on inter-cluster
    
    communication speeds)
    
    Aware (**CA**)  

Co-Allocation Policies (3)

WF

Components

Clusters

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

FCM

Component

24

C1 (16)  C2 (16)  C3 (16)
Experimental Results (1)

- DAS2
- MPICH-G2 + Globus
- Communication-intensive applications
- Workload 1: low load
- Workload 2: high load
- Background load: 15-20%

Experimental Results (2)

- DAS3
- OpenMPI + DRMAA
- Myri-10G vs. GbE
- FCM vs. CA
- Communication-intensive applications

<table>
<thead>
<tr>
<th>number of clusters combined</th>
<th>Myri-10G</th>
<th>GbE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>average execution time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCM [w/o Delft]</td>
</tr>
<tr>
<td>CA</td>
</tr>
<tr>
<td>FCM [with Delft]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>average job response time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait time</td>
</tr>
<tr>
<td>execution time</td>
</tr>
<tr>
<td>FCM [w/o Delft]</td>
</tr>
<tr>
<td>CA</td>
</tr>
<tr>
<td>FCM [with Delft]</td>
</tr>
<tr>
<td>CA</td>
</tr>
</tbody>
</table>
Support for Malleable Applications

• Background

• System Design

• Malleability Management Policies

• Experimental Results
Parallel Application Types

- **Rigid**
  - Requires a fixed number of processors

- **Moldable**
  - The number of processors can be adapted only at the start of the execution

- **Malleable**
  - Number of assigned processors can be changed during runtime (i.e., grow/shrink)

See. D. G. Feitelson and L. Rudolph, “Toward convergence in job schedulers for parallel supercomputers,” in *JSPP'96*
Aspects of Supporting Malleability

• Specification of malleable jobs
  • A minimum and a maximum number of processors

• Initiative of change (grow/shrink requests)
  • Application initiated
  • Scheduler initiated

• The obligation to change
  • Voluntary
  • Mandatory
System Interaction (1)

- **DYNACO** is a framework for building dynamically adaptable applications; more on [http://dynaco.gforge.inria.fr](http://dynaco.gforge.inria.fr)

  
  - Malleable Policies:
    - Favour Previously Started Malleable Applications (FPSMA)
      - grow the earliest started
      - shrink the latest started
    - Equi–Grow & Shrink (EGS)
      - Distribute total amount of growing/shrinking equally among running jobs
Phase 1: job placement

Scheduler

placement queue

new submission

Runner

retry

Launch Job

Placement Policies: Place Job (WF, CF, CM)

Malleable Job

clusters

13-5-2009
System Interaction (3)

Event: Processors became available

Malleability Manager: PRA/PWA

Policies: Grow/Shrink (FPSMA, EGS)

Scheduler

Runner

grow/shrink
grown/shrunk

clusters

Phase 2: manage malleable applications
Experimental Results

- DAS3
- DYNACO-based MPI Applications
- Wm: Malleable jobs only
- Wmr: Mixed Workload: 50% Malleable jobs 50% Rigid Jobs

Wm resulted in better performance
Similar performance results for FPSMA and EGS policies

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
  • System contention

• Run them as **Cycle Scavenging Applications !!**
  • Sets priority classes implicitly
  • No need for personal monitoring
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
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 have 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

Scheduler

CS-Runner

User submits PSA(s)

JDF

Application Level Scheduling:
• Pull-based approach
• Shrinkage policy

Head Node

KCM

Node

Launcher

Launchers

Clusters

monitors/informs idle/demanded resources

registers

grow/shrink messages

submits launchers

deploys, monitors, and preempts tasks
Cycle Scavenging Policies

1. Equipartition-All (grid-wide basis)
Cycle Scavenging Policies

2. Equipartition-PerSite (per cluster)
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


Equi-PerSite is fair and superior to Equi-All
Support for Workflows in Koala

• In Progress...
• We support **Pegasus-Chimera** job description language
• Experience the WF-Runner in the **hands-on** session

• Single-Site Policy (static decision-taking):
  • Map a workflow to the least loaded cluster

• Multiple-Site Policy (dynamic decision-taking):
  • Submit each task to the least loaded cluster
Conclusion

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

• **Future Work:**
  • More workflow policies
  • A decentralized P2P KOALA
  • Cloud resource management
Questions?

More Information:

• KOALA: www.st.ewi.tudelft.nl/koala
• DYNACO: http://dynaco.gforge.inria.fr
• DAS3: www.cs.vu.nl/das3

Publications

• see PDS publication database at www.pds.ewi.tudelft.nl