ALiCE – A Java-based Grid Computing System

Assoc Professor Teo Yong Meng
Department of Computer Science
National University of Singapore

teoym@comp.nus.edu.sg
http://www.comp.nus.edu.sg/~teoym
Alessandro Volta in Paris in 1801 inside French National Institute shows the battery while in the presence of Napoleon I

What? This is a mad man...

....and in the future, I imagine a worldwide Power (Electrical) Grid......
Overview

- grid computing
- potential benefits
- ALiCE
  - main features
  - execution model
  - Implementation
  - programming model
  - some ALiCE examples

- related works
  - current status
  - limitations
  - examples
Grid Computing
Current Internet Technology

• Internet was conceived in the late 1960s as a peer-to-peer system to share computing resources.

• Current Internet technologies address communication and information exchange among computers but not the coordinated use of resources at multiple sites for computation.

• The aim of grid computing is to exploit unused latent computing capabilities of idle resources (hardware, software, etc.) for solving large-scale compute intensive problems.
Grid Computing

- High performance computing techniques that seek to efficiently coordinate the use of geographically dispersed compute resources linked by networks to solve large, complex problems.

- A computational grid must integrate heterogeneous compute resources with dynamic availability, connected by an unreliable network.
Potential Benefits of Grid Computing

1. sharing but more importantly aggregation of resources and leveraging on resources you don’t own;
2. reduce business cost by maximizing the utilization (and return of investment) of existing corporate computing resources through better utilization (and selling of “idle” resources)
3. meet the need for short-term supercomputing power without the need for costly investment - computational capabilities;
4. provide remote access to expensive computational resources and equipments, and hard to distribute, large and proprietary data sets;
5. computing on demand reduces business costs

reduces business costs, capabilities and scalability, fault tolerance, .......
ALiCE Grid Computing
ALiCE (Adaptive and scalable Internet-based Computing Engine)

Client/Server Model

Grid Model

ALiCE Brokered Grid Model
Main Objectives

• to support computing on demand and resource aggregation (resource sharing)

• to support the development and execution of generic grid applications, i.e., run applications over a geographically distributed, heterogeneous collection of resources (computers, networks, etc.)

• a middleware that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware and operating systems
Main Features

- job-parallelism to maximize **throughput** and (Java) object-parallelism to maximize **performance**

- **distributed load-balancing algorithm**
  - **application-driven** distributed load balancing;
  - task replications for fault-tolerant and to meet performance deadline

- differentiated levels of security (code, data and result) at varying costs

- **ALiCE** is implemented using ubiquitous and easy-to-use Java, Java Jini/JavaSpaces technologies

- adaptive parallelism, cross-platform portability, extensibility and scalability
ALiCE Producer-Consumer Model

- **consumers (clients)**
  - execute jobs for solving problem of varying size and complexity
  - benefit by selecting and aggregating resources wisely
  - tradeoff timeframe vs cost - minimize IT expenses

- **producers (servers)**
  - contribute “idle” resource for executing consumer jobs
  - benefit by maximizing resource utilization
  - tradeoff local requirements vs market opportunity - maximize return on IT investment

- **resource broker (agency)**
  - facilitates and automate the process of resource sharing
  - coordinates the allocation of producer resources to match consumer required performance requirement and cost
  - regulates demand-and-supply
**ALiCE Architecture (1)**

**Consumers (C)**
- interface to users
- launch point for applications
- collection point for results (visualization)

**Resource Broker (RB)**
- authentication
- application execution control
- resource management
  - scheduling
  - load balancing
- ...

**Producers (P)**
- provide computing power
- executes tasks
ALiCE Architecture (2)

Resource Broker
- Scheduler
- Accounting
- Performance Monitor
- Security Manager
- Pricing Manager

Producer
- Registration
- WorkEngine
- Local Scheduler
- Performance Measurement

Consumer
- Registration
- Task Submission
- Result Retrieval

Task Pool

25 October 2001
Teo Yong Meng, NUS
Types of Applications Supported

1. Sequential Jobs (parametric computation)
   - supports single-tasking programs with well-defined methods like main() or run()

2. Parallel Jobs - Object-level Parallelism
   - supports various parallel programming models via programming templates
   - allows task and result objects to be exchanged between consumers and producers through resource broker
ALiCE Consumer GUI

Control Panel

Resource Broker Info

Task Input

User Requirements

Messages

25 October 2001 Teo Yong Meng, NUS
ALiCE Programming Models

- Master-slave, shared-memory, data parallel, etc. through a library of programming templates
- Transparency & programmability – hides JavaSpaces
- TaskGenerator-ResultCollector model:
  - Task Generator (TG)
  - Result Collector (RC)
Programming Templates

ALiCE programming template includes the following specifications:

1. Task Generator, Result Collector, and Task Classes

2. Method used to send Task objects to ALiCE

3. Result object used to store result information

4. Method used to retrieve the Result objects from ALiCE.
Task Generator Template

// Task Generator Template
import alice.application.*; // import the templates

public class CLASSNAME extends TaskGenerator {

    // place your variables here

    // Constructor
    public CLASSNAME(){};

    // The no parameter constructor is a MUST.

    public void init() {
        // place your initialization code here
    } // init()

    // This is where the tasks are generated
    // Usually tasks are generated in a loop,
    // and in this loop each task is sent for
    // processing by calling the
    // "public void process(Task t)" method

    public void generateTasks() {
        // This is where the tasks are generated
        // Usually tasks are generated in a loop,
        // and in this loop each task is sent for
        // processing by calling the
        // "public void process(Task t)" method

    } // generateTasks()

    public static void main(String args[]) {
        CLASSNAME m = new CLASSNAME();
        m.init();
        m.generateTasks();
    }

} // end class
// Template for ResultCollector
//
import alice.application.*; // import the templates

public class CLASSNAME extends ResultCollector {

    // place your variables here
    //

    public static void main(String args[]) {

        CLASSNAME MV = new CLASSNAME();
        MV.init();
        MV.collectAllResults();
    }

    // the no argument constructor MUST exist
    public CLASSNAME() {
    }

    public void init() {
        // place your init codes here
        //
    }

    public void collectAllResults() {

        // Here is the result handling code.
        // Usually result handling involves a loop
        // that repeatedly calls the collectResult()
        // method of the ResultCollector superclass.
        //
        // This method returns a Result Object.
        //
        // The contents of this Result Object can be
        // inspected for result handling/processing.
    }

}
// Template for Task

import alice.application.*; // import the templates

public class CLASSNAME implements Task {
    // place your variables here
    //
    public CLASSNAME() {}  
    public Result execute() {
        // This is where you do your calculation
        // The results are stored in the Result class
        // which functions as a datastructure
        // with which you can store results of any Object type
    }

    public String toString() {
        // returns a String that can be used to ID your task
    }
}
ALiCE Implementation (1)

Issues:
- How to send user application from Consumer to RB?
- How to initiate the user application at RB with tasks executing at Producers?
- How to distribute code and data?
- How to collect results?
- How to coordinate communication between consumers, producers and RB?

Core Mechanisms
- JavaSpaces
- Reflection API & RMI
- ClassLoader
- Job Submission
- Job Execution
- Result Collection & Visualization
ALiCE IMPLEMENTATION (2)

Resource Broker
- Java SDK 1.3
- Java Jini 1.1/JavaSpaces
- RMI

Task Pool
- Java SDK 1.3
- Java Jini 1.1/JavaSpaces
- Java Reflection API
- RMI
- Swing

Consumer
- Java SDK 1.3
- Java Jini 1.1/JavaSpaces
- RMI
- Swing

Producer
JavaSpaces

- **JavaSpaces**
  - a tuple-space for Java Objects
  - a communication medium for distributed Java applications

- **JavaSpaces Server** (resides at RB)
  is the communication medium between Consumers, Producers and RB

- However, JavaSpaces is not designed for dynamic code distribution and execution.
Reflection API & RMI ClassLoader

- **Java reflection API**
  - Represents Classes, Objects in the Java Virtual Machine
  - Allows invocation of methods whose name and Class are known only at runtime.

- **RMI Classloader**
  - A mechanism that provides the dynamic class loading of the RMI runtime system
  - Allows the transport and loading of classes from remote hosts.

We augment JavaSpaceS with these two mechanisms to provide a solution for **dynamic code distribution and execution**.
Some Applications-Divers

1. Distributed HPC (Supercomputing)
   - computational science - a massively, scalable, distributed, virtual supercomputer

2. High-throughput Computing
   - large-scale simulation/chip design/complex financial analysis & parameter studies

3. Data-intensive Computing
   - data mining, drug design, particle physics (CERN),....
Some ALiCE Applications

- ray tracing
- Mandelbrot set
- georectification of satellite images
ALiCE - Distributed Mandelbrot Set
Tropical Red Tide Monitoring

The Seastar satellite which carries the SeaWiFS sensor.

L0 Data
Raw data from SeaWiFS sensor.

L1A Data
Reconstructed, unprocessed instrument data at full-resolution.

L1B Data
Calibration corrected radiance data.

L2 Data
Geophysical data.

L3 Data
Space-time grid mapped geophysical data.

Scientific or geographical usage of data

The Seastar satellite which carries the SeaWiFS sensor.

L1A Georectified Data
Earth geometry corrected data.

L1B Georectified Data
Earth geometry corrected data.

L2 Georectified Data
Earth geometry corrected data.

L3 Data
Space-time grid mapped geophysical data.
Georectification of Satellite Images

Before Georectification

After Georectification
ALICE - Georectification of Images
Related Works

- current status
- limitations
- examples
# Related Works

<table>
<thead>
<tr>
<th>Grid System</th>
<th>Comm. Technology</th>
<th>Language Supported</th>
<th>Platforms</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legion - Virginia</td>
<td>Sockets</td>
<td>C++, Mentat, Fortran, PVM, MPI</td>
<td>Solaris (SPARC), SGI, linux (x86, alpha, dec), HPUX, RS6000</td>
<td>Tree Structure (global root node)</td>
</tr>
<tr>
<td>GLOBE - Vrije</td>
<td>Java IDL</td>
<td>Java, C</td>
<td>Win NT, Unix</td>
<td>Tree Structure (global root node)</td>
</tr>
<tr>
<td>Javelin - UCSB</td>
<td>Java Applets, Java RMI</td>
<td>Java</td>
<td>Any Java enabled browser</td>
<td>Three-Tier Clients, Brokers, Hosts</td>
</tr>
<tr>
<td>Calpyso - NYU/Arizona</td>
<td>Sockets</td>
<td>C++</td>
<td>Solaris, Linux, Win NT</td>
<td>Three-Tier Clients, Brokers, Hosts</td>
</tr>
<tr>
<td>Charlotte - NYU</td>
<td>Java Applets, Java RMI</td>
<td>Java</td>
<td>Any Java enabled browser</td>
<td>Three-Tier Clients, Brokers, Hosts</td>
</tr>
<tr>
<td>Knitting Factory</td>
<td>Java Applets, Java RMI</td>
<td>Java</td>
<td>Any Java enabled browser</td>
<td>Three-Tier Clients, Brokers, Hosts</td>
</tr>
<tr>
<td>Condor - Wisconsin</td>
<td>RPC</td>
<td>C</td>
<td>Win NT, Unix</td>
<td>Client-Server</td>
</tr>
<tr>
<td>Nile - Cornell</td>
<td>Java, CORBA</td>
<td>Java</td>
<td>Any Java compatible platform</td>
<td>unknown</td>
</tr>
<tr>
<td>Bayanihan - MIT</td>
<td>HORB (Java RMI equivalent) using Java Applets</td>
<td>Java</td>
<td>Any Java enabled browser</td>
<td>Client-Server Clients are resources only, they cannot submit jobs</td>
</tr>
<tr>
<td>ALiCE</td>
<td><strong>JavaSpaces</strong> Java RMI</td>
<td>Java</td>
<td>Any Java compatible platform</td>
<td>Three-Tier Clients can be a resource provider, a resource consumer, and both</td>
</tr>
</tbody>
</table>
Current Status

- who contribute resources?
  - Volunteers

- why?
  - Motivated by the public good, fun, challenge, fame, charismatic applications, etc.

- public good projects such as SETI@HOME, genome@home, folding@home, etc.
Some Limitations

- specialized applications and clients
- cannot process \textit{generic} data or handle generic computational tasks
- clients must be \textit{upgraded} to handle new tasks

We need a ubiquitous, portable, infrastructure that can handle \textit{generic} computational tasks and data.
SETI@Home
The Search for Extraterrestrial Intelligence at Home

Data Analysis
Searching for Pulses / Triplets: 69%
Doppler drift rate: 44.0076 Hz/sec, Resolution: 1.197 Hz
Best Pulse: power 1.00, period 0.364 sec, score 1.01

Data Info
- From: 13 hr 29' 34" RA, +12 deg 24' 53" Dec
- Recorded on: Mon Dec 11 11:55:46 2000 GMT
- Source: Arecibo Radio Observatory
- Base Frequency: 1.41906250Hz

User Info
- Name: Pureman
- Data units completed: 1185
- Total computer time: 28662 hr 39 min 32.1 sec

25 October 2001
Teo Yong Meng, NUS
25 October 2001

Teo Yong Meng, NUS
Pande Group – Stanford University

- genome@home
- folding@home
Intel Philanthropic
Peer-to-peer Program

- Partners: American Cancer Society, National Foundation for Cancer Research, University of Oxford, United Devices

Leukemia Drug-Optimization Program
Hidden Markov Model – Pattern Matching in Genetic Sequences

This computer is running the HMMER application developed at Washington University in St. Louis. This program uses the Hidden Markov Model (HMM) statistical tool to compare large genetic sequences against protein domains. At left, the letters represent different amino acids. The brighter the color on the letters, the better the genetic match. The "Residue start" and "Residue end" denote the start and end of the matching portion. Identifying matching sequences helps researchers develop screening tests and drugs for the prevention and cure of diseases.
Acknowledgements

Collaborator: Centre for Remote Imaging, Sensing and Processing (NUS)

Bioinformatics company (negotiating)

Current Students:
Johan Prawira Gozali, Ng Yean Seen, Low Seo Chin, Zheng Yudong, Ng Yew Kwong

Past Students: Lee Chwan Ren, Dr Wang Chen
The Future of P2P computing

Thank you.
Questions & Answers