ALiCE: A Lightweight Grid Middleware

Teo Yong Meng

Department of Computer Science
National University of Singapore

teoym@comp.nus.edu.sg

http://www.comp.nus.edu.sg/~teoym
Cost of Idle Computing Cycles

Desktop Processor Utilisation

<table>
<thead>
<tr>
<th></th>
<th>$ / processor (desktop)</th>
<th>$ / used</th>
<th>$ / used processor</th>
<th>cost of unused cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>one desktop</td>
<td>$1200</td>
<td>$300</td>
<td>$150</td>
<td>$1050</td>
</tr>
<tr>
<td>1000 desktops</td>
<td>$1,200,000</td>
<td>$300,000</td>
<td>$150,000</td>
<td>$1,050,000</td>
</tr>
</tbody>
</table>

Source: Adapted from Internet Infrastructure & Services by Bear, Stearns & Co., May 2001. Based on IDA tender price, the cost of a Pentium PC desktop is estimated to be $1200.

Assumptions:

1. Desktop utilization is 25%; 8 hrs/24 hrs = 33%; factoring in lunch, restroom, etc. a desktop can be idle up to 90%
2. When a processor is in used, assume a peak utilization of 50%
Outline

- Grid computing overview
- Overview of Globus
- ALiCE Middleware
  - Key features
  - Producer-consumer model
  - Template-based grid programming
  - ALiCE applications
  - ALiCE vs Globus
- Supercomputer, Physical and Virtual Cluster
Grid Computing (1)

Client/Server Model

Grid Model

The Service Grid
Grid Computing (2)

• Flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resource
  From “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”

• Enable communities (“virtual organizations”) to share geographically distributed resources as they pursue common goals -- assuming the absence of...
  - central location,
  - central control,
  - omniscience,
  - existing trust relationships
Grid Computing (3)

- **Resource sharing**
  - Computers, storage, sensors, networks, databases, ...
  - Sharing always conditional: issues of trust, policy, negotiation, payment, ...

- **Coordinated problem solving**
  - Beyond client-server: distributed data analysis, computation, collaboration, ...

- **Dynamic, multi-institutional virtual orgs**
  - Community overlays on classic org structures
  - Large or small, static or dynamic
Grid Computing (4)

Advantages
- sharing and aggregation of resources
- leveraging on resources you don’t own
- “computing on demand”
  - focus on business rather than technology
  - reduce business costs
- remote access to expensive resources (proprietary data sets,..)
- capability and scalability
- fault tolerance
- ..... 

Challenges
- heterogeneity
- distributed ownership
- dynamic behavior – internet based on best effort packet delivery
- security
- ease of use
- .....
Design Complexities

Too many objectives

Not enough principles!

Grid System

availability
security
consistency
atomicity
mobility
scalability
performance
maintainability
shared data
decentralization
ease of use
Three Main Obstacles in Grid Computing

1) New approaches to problem solving
   - Data Grids, distributed computing, peer-to-peer, collaboration grids, ...

2) Structuring and writing programs
   - Abstractions, tools Programming Problem

3) Enabling resource sharing across distinct institutions Systems Problem
   - Resource discovery, access, reservation, allocation; authentication, authorization, policy; communication; fault detection and notification; ...

26 May 2002 Teo Yong Meng, NUS
Globus Layered Grid Architecture

"Coordinating multiple resources": ubiquitous infrastructure services, app-specific distributed services

"Sharing single resources": negotiating access, controlling use

"Talking to things": communication (Internet protocols) & security

"Controlling things locally": Access to, & control of, resources
Globus Layered Grid Architecture

Applications

High-level Services and Tools

Core Services

Local Services

DRM
Cactus
GASS
GridFTP
Condor
MPI
Metacomputing Directory Service
Globus Security Interface
Replica Catalog
GRAM
I/O
TCP
UDP
LSF
PBS
NQE
Linux
AIX
Solaris
globusrun
PUNCH
Nimrod/G
Condor-G
Grid Status

High-level Services and Tools include:
- Globus
- GridFTP
- Grid Security Interface
- Replica Catalog
- GRAM
- I/O

Core Services include:
- Cactus
- Condor-G
- Condor
- GridFTP
- LSF
- Solaris

Local Services include:
- Condor
- GridFTP
- LSF
- Linux
- MPI
- NQE
- PBS
- TCP
- UDP
- AIX
- Solaris

Applications include:
- DRM
- Cactus
- MPI
- globusrun
- PUNCH
- Nimrod/G
- Condor-G
Globus Toolkit

• Software toolkit
  - defines a set of services (grid protocols and APIs)
  - (partially) implemented as of a collection of tools

• Focus is on inter-domain issues, not clustering
  - supports collaborative resource use spanning multiple organizations
  - integrates with intra-domain services
• Key features
• Producer-consumer model
• Template-based programming
• ALiCE Applications
• Globus vs ALiCE
• Supercomputer, physical and virtual grid
What is ALiCE (Adaptive and scalable Internet-based Computing Engine)?

Client/Server Model

Grid Model

ALiCE
Brokered Grid Model
ALiCE (Adaptive and scalable internet-based Computing Engine)

- Support for development and deployment of grid applications
- Template-based programming to mask complexity of grid infrastructure
- Job-parallelism to maximize throughput
- (Java) object-parallelism to maximize performance
- Distributed load-balancing algorithm
- Task replications for fault-tolerant and meeting performance deadline
- Differentiated levels of security (code, data and result) at varying costs
- Implemented in Java and Java Jini™/JavaSpaces™
ALiCE Producer-Consumer Model

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 IMPLEMENTATION

Resource Broker

- Java SDK 1.3
- Java Jini 1.1/JavaSpaces
- Java Reflection API
- Swing

Task Pool

Consumer

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

Producer

- Java SDK 1.3
- Java Jini 1.1/JavaSpaces
- Java Reflection API
- Swing
ALiCE Consumer GUI
ALiCE Producer GUI

Control Panel

Task Information

Performance

Messages

26 May 2002
Teo Yong Meng, NUS
1. JobLauncher sends jar file via TCP

2. TaskManager receives jar file and invokes RMI ClassLoader to execute it.

3. Allocator component sends Task to Producer via JavaSpace

4. Producer creates a WorkEngine to execute it

5. WorkEngine completes execution and sends untagged result back to Resource Broker via JavaSpace

6. Tagged Result object with consumer address

7. Results received by ResultsListener and stored in ResultsPool

8. User-defined visualizer will collect results from the pool when it needs to.
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
## Template-based Programming

<table>
<thead>
<tr>
<th>Template</th>
<th>Function</th>
</tr>
</thead>
</table>
| TaskGenerator       | • Invoked at resource broker   
                      | • Method to send tasks to producer                                     |
| ResultCollector     | • Visualizer to be invoke at consumer  
                      | • Method to retrieve results                                          |
| Task                | • Specify functions to execute at producer  
                      | • Return a Result object                                               |
| Result              | • Interface for producer to instantiate and return result                |
Job Execution

TaskGenerator + Task class file (jar file)

Resource Broker

Producer machines

Result

process()

Task

... 

Task

collectResult()
// 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()
    }

    /**
     * generateTasks() - generates tasks
     */

    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()

    }  // 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.
    }
}
Task Template

// 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
    }
}

26 May 2002 Teo Yong Meng, NUS
ALiCE Applications

- Georectification of satellite images (CRISP)
- Protein alignment and matching (BII)
- Primer Search in Chromosome Sequences (Nanyang Polytechnics)
- Distributed ray tracing
- Distributed equation solver
- Mandelbrot set
- N-body problem

26 May 2002
Teo Yong Meng, NUS
Georectification of Satellite Images

Original Image

Image after Georectification

Setup parameters
Select GCPs

Transform co-ordinates (T)
Resample image (R)
Visualise image (V)

Add gridlines & boundaries
Finalise image

Setup | Execution | Termination

26 May 2002
Teo Yong Meng, NUS
public class swTask implements Task {
    public Result execute() {
        // ... initialize data structures
        for (int j=0;j<chunkHeight;j++) {
            for (int i=0;i<chunkWidth;i++) {
                // perform coordinate transformation & image resampling */
                return result;
            }
        }
        // end execute()
        // ... other methods
        //
    }// end class

    public class swGenerator extends TaskGenerator { public void generateTasks() {
        // ... initialize variables
        for (int h=0;h<chunkHeight;h++) {
            for (int w=0;w<chunkWidth;w++) {
                swTask _swTask=new swTask(); // create task
                process(_swTask); // send task to Res. Broker
                taskGenerated++; //
            }
        }
        // end generateTask()
        // ... other methods
        //
    }// end class

    public class swTaskListener extends ResultCollector {
        // ... initialize data structure

        public void run() {
            while (resultsCollected<resultsExpected) {
                swTask result=((swTask)swGUI.collectResult()); // collect results
                vResults.add(result); // store result for visualisation
                // ... datastructure maintenance code
            }
            // end run()
            // ... other methods
            //
        }// end class

    26 May 2002  Teo Yong Meng, NUS  30
ALiCE Applications

Protein Alignment and Matching – 50 MB chromosome database, swissprot (NCBI BLAST d/b server), P2, 450MHz, 256MB memory

- Sequential ~ 2 hours
- ALiCE with 8 producers, 4000 sequences/task ~1400 seconds

N-body problem (n=20,000 bodies) – predicting the motion of astronomical bodies in space. - Sequential - 9428 seconds

- ALiCE: 2 producers = 1109 sec, 8 produces = 457 sec
Globus vs ALiCE

- **Globus**
  - is a Grid Toolkit (provides set of services)
  - Is an open system (users can develop higher-level services on top of basic services)
    - Adv: modularity & reusability of services
    - Dis: Grid Infrastructure setup is complex
    - app development/deployment is complex
  - is largely platform dependent (mostly UNIX)
  - Resource Management (via GRAM) at Job Level
Globus vs ALiCE

• ALiCE
  - User-Oriented Grid Computing Engine
    • Integrates all services (basic+higher level) to facilitate:
      - Ease of installation, deployment and administration
      - Ease of Grid App Development/Deployment using object programming template

    • Platform Independent (core services implemented in Java)

    • Resource Management at Object-level (fine-grain control)
Supercomputers

June 29, 2000 IBM ASCI White
- 8192 RS/6000 processors, 12.3 TFLOPS
- 6 TB memory, 160 TB disk storage
- US$110m, 106 tons, 28 tractor-trailer trucks

April 20, 2002 NY Times - Japanese Computer is World’s Fastest
- NEC
- US$350m, occupies 4 tennis-court
- 640 specialized nodes, 5104 processors
- achieved 35.6 TFLOPS versus 7 TFLOPS in ASCI White
Cost of a Supercomputing, a Physical Cluster and a Virtual Grid of 100,000 PCs

- IBM ASCI White (2000) - US$110m
- NEC (2002) - US$350m
- Physical cluster of 100K Intel P4 ~ US$200m + US$13m (electricity)
- Virtual cluster - cost is distributed and absorbed by PC owners
Acknowledgements

Collaborators:
- Centre for Remote Imaging, Sensing and Processing (CRISP)
- BioInformatics Institute
- Nanyang Polytechnic (School of Life Sciences)
- The Royal Institute of Technology, Sweden

Acknowledgement: Sun Microsystems
Thank you.

Questions & Answers

Prediction is even harder in a field characterized by exponential progress

“I think there is a world market for maybe five computers” — Thomas J. Watson, founder and Chairman of IBM, 1943