The ProActive Parallel Suite

Fabrice Huet, INRIA-University of Nice

Joint work with ActiveEon
Outline

• Overview of ProActive Parallel Suite
  • Active objects
  • GCM Deployment

• ProActive Scheduler

• Resource Manager

• Multi-Active Objects
## PROGRAMMING

### Java Parallel Toolkit
Accelerate & Scale your most demanding applications with Multi-Core and Distributed Parallel Computing:

- Portability, Fault-tolerance, File transfer, SOA, Security
- Master Worker, Monte Carlo, Legacy Wrapping, SPMD, File Split & Merge, Task Flow
- Asynchrony, groups, futures
- Monitoring & Optimizing GUI

## RESOURCING

### Desktop, Cluster, Grid, Cloud Resource Manager
Leverage existing infrastructures:

- Private Cloud, Virtualization (VMware, KVM, Xen)
- Dynamic Resource Acquisition (including EC2, NVIDIA GPU)
- Deploy applications with ssh, rsh, sshGSI
- Schedule with PBS, LSF, SGE, Prun, EGEE gLite, IBM LL
- Windows and Linux Agents

## SCHEDULING

### Multi-Platform Job Scheduler
Schedule parallel jobs on all company resources and on external clouds:

- Fault-tolerance, Security
- File Transfer
- Java and XML Taskflows
- Java or native applications, scripts (Ruby, Python, JavaScript), Matlab, Scilab, MPI, ProActive Programming
Programming

ACTIVE OBJECTS
Active Objects

With ProActive, he gets …

Developer writes
Object A

new A(...) newActive(A,..)
An active object

- Is coarse-grained structuring entities (subsystems)
- Has one thread and a request queue
- Owns passive objects (Java Objects, no thread)
  - No shared passive objects, parameters are deep-copied
- Has asynchronous communication (local / remote)
- Is insensitive to application deployment
Active object

An active object is composed of several objects:

• The object being activated: Active Object (1)
• A set of standard Java objects
• A single thread (2)
• The queue of pending requests (3)
Active objects

```java
A ag = newActive ("A", [...], Node);
V v1 = ag.foo (param);
V v2 = ag.bar (param);
...
v1.bar(); //Wait-By-Necessity
```
### Active objects

- A `ag = newActive ("A", [...], Node)`
- V `v1 = ag.foo (param);`
- V `v2 = ag.bar (param);`
- ...
- `v1.bar(); //Wait-By-Necessity`
Wait by necessity

• A call on an active object consists in 2 steps
  • A query: name of the method, parameters…
  • A Reply: the result of the method call

• A query returns a **Future** object which is a placeholder for the result

• The callee will update the **Future** when the result is available

• The caller can continue its execution event if the **Future** has not been updated

```java
foo ()
{
    AnyObject r = a.process();
    //do other things
    ...
    r.toString();
} //will block if not available

AnyObject process()
{
    //perform long computation
    return result;
}
```
Explicit Synchronizations

\[
A \text{ ag = newActive (“A”, [...], VirtualNode)} \\
V \text{ v = ag.foo(param);} \\
\ldots \\
v.bar(); //Wait-by-necessity
\]

Explicit Synchronization:

\[
- - \text{PAFuture.isAwaited \(v\);} \quad \text{// Test if available} \\
- - \text{waitFor \(v\);} \quad \text{// Wait until available}
\]

Collections of Futures:

\[
- - \text{waitForAll \(\text{Collection}\);} \quad \text{// Wait All} \\
- - \text{waitForAny \(\text{Collection}\);} \quad \text{// Get First}
\]
Call between Objects: Parameter passing: Copy of Java Objects

(Deep) Copies evolve independently -- No consistency
Call between Objects: Parameter Passing: Active Objects

Object passed by Deep Copy - Active Object by Reference
Wait-By-Necessity: First Class Futures
Futures are Global Single-Assignment Variables

\[
V = b.\text{bar}() \\
c.\text{gee}(V)
\]
ProActive: Reuse and seamless

Polymorphism between standard and active objects
- Type compatibility for classes (and not only interfaces)
- Needed and done for the future objects also

Wait-by-necessity: inter-object synchronization
- Systematic, implicit and transparent futures
- Ease the programming of synchronizations, and the reuse of routines
Programming

GROUPS
ProActive Groups

- Manipulate groups of Active Objects, in a simple and typed manner:
  - Typed and polymorphic Groups of local and remote objects
  - Dynamic generation of group of results
  - Language centric, Dot notation

- Be able to express high-level collective communications (like in MPI):
  - broadcast,
  - scatter, gather,
  - all to all

\[
A \text{ ag} = (A) \text{ PAGroup.newGroup} (\langle A \rangle, \{\{p1\}, \ldots\}, \{\text{Nodes}, \ldots\});
\]

\[
V \ v = \text{ ag.foo} (\text{param});
\]

\[
v.\text{bar}();
\]
ProActive Groups

- **Group Members**
  - Active Objects
  - POJO
  - Group Objects

- **Hierarchical Groups**

- **Based on the ProActive communication mechanism**
  - Replication of N ‘single’ communications
  - Parallel calls within a group (latency hiding)

- **Polymorphism**
  - Group typed with member’s type
Creating AO and Groups

- A \( ag = \text{PAGroup.newGroup} \) ("A", [...], Node[])
- V \( v = ag.foo(param) \);
- ...
- \( v.bar() \); //Wait-by-necessity
Typed Group as Result of Group Communication

Ranking Property:
- Dynamically built and updated
  - \( B \text{ groupB} = \text{groupA}.\text{foo}(); \)
- Ranking property: order of result group members = order of called group members

Explicit Group Synchronization Primitive:
- Explicit wait
  - \( \text{ProActiveGroup.waitOne(groupB);} \)
  - \( \text{ProActiveGroup.waitAll(groupB);} \)
- Predicates
  - noneArrived
  - kArrived
  - allArrived, ...
Programming

GCM DEPLOYMENT
Abstract Deployment Model

Problem

Difficulties and lack of flexibility in deployment
Avoid scripting for configuration, getting nodes, connecting…

A key principle: Virtual Node (VN)

Abstract Away from source code:
- Machines names
- Creation/Connection Protocols
- Lookup and Registry Protocols

Interface with various protocols and infrastructures:
- Cluster: LSF, PBS, SGE, OAR and PRUN (custom protocols)
- Intranet P2P, LAN: intranet protocols: rsh, rlogin, ssh
- Grid: Globus, Web services, ssh, gsissh
Resource Virtualization

Runtime structured entities: 1 VN --> n Nodes in m JVMs on k Hosts
Deployment Descriptor- Examples

VirtualNode:
  *myNode*
Mapping:
  *myNode* $\rightarrow$ VM1, VM2
Infrastructure:
  VM1 $\rightarrow$ Local VM
  VM2 $\rightarrow$ SSH host1
  then Local VM

VirtualNode:
  *myNode*
Mapping:
  *myNode* $\rightarrow$ VM1
Infrastructure:
  VM1 $\rightarrow$ SSH frontend
  then pbsProcess
  pbsProcess $\rightarrow$ PBS 10 nodes Local VM
Details

• Requires two XML files
  • GCMA : Application needs (Name of executable, path, variables…)
  • GCMD : Description of the infrastructure (Address of machines, protocols to use…)

• API to deploy from inside the application
  • Non-blocking

```java
pad = PAGCMDeployment.loadApplicationDescriptor(new File(descriptor));
pad.startDeployment();
pad.waitReady();
GCMVirtualNode vn = pad.getVirtualNodes().values().iterator().next();
PAActiveObject.newActive(……., vn.getANode());
```
Status of GCM deployment

• Considered as a legacy tool
• Still working but not supported
  • Some protocols certainly don’t work anymore
• Fundamental issues
  • Each application has to deploy its nodes (slow/costly)
  • Cannot use a VN until all nodes are available
• Minor issues
  • Difficult to add new protocols, especially complex one like EC2
  • Lack of flexibility of the XML files
• But still useful for quick’n’dirty runs!
Conclusion on ProActive Programming

• Simple and Uniform programming model
• Rich High Level API
• Write once, deploy everywhere
• Let the developer concentrate on his code, not on the code distribution
• Easy to install, test, validate on any network
**Programming**

Java Parallel Toolkit
Accelerate & Scale your most demanding applications with Multi-Core and Distributed Parallel Computing:
- Portability, Fault-tolerance, File transfer, SOA, Security
- Master Worker, Monte Carlo, Legacy Wrapping, SPMD, File Split & Merge, Task Flow
- Asynchrony, groups, futures
- Monitoring & Optimizing GUI

**Resourcing**

Desktop, Cluster, Grid, Cloud Resource Manager
Leverage existing infrastructures:
- Private Cloud, Virtualization (VMware, KVM, Xen)
- Dynamic Resource Acquisition (including EC2, NVIDIA GPU)
- Deploy applications with ssh, rsh, sshGSI
- Schedule with PBS, LSF, SGE, Prun, EGEE gLite, IBM LL
- Windows and Linux Agents

**Scheduling**

Multi-Platform Job Scheduler
Schedule parallel jobs on all company resources and on external clouds:
- Fault-tolerance, Security
- File Transfer
- Java and XML Taskflows
- Java or native applications, scripts (Ruby, Python, JavaScript), Matlab, Scilab, MPI, ProActive Programming
Scheduling

PROACTIVE SCHEDULER
Rational

• Allowing non grid-proficient user to use a Grid/Cluster/Cloud

• Target users who
  • Want to start their code within their environment (their files, libraries…)
  • Don’t really care where their code is running
  • Have different requirements (from “I need a 3Ghz CPU” to “I need maple”)

• The ProActive scheduler is a simple scheduler
  • Written in Java
  • With/Without GUI
  • Handles jobs submitted by users
  • Run them on any resource available

• Designed to work in a trusted environment
In a nutshell

• Resources (Java Virtual Machines) register into the resource manager
  • Only a JVM + jar file needed, no installation

• Jobs are made of tasks
  • Can specify dependency
  • Tasks are java classes or any native application

• The scheduling is very basic
  • Find a free resources
  • Find the next task to be executed
  • Check whether the resource can run the task
  • Run the task
Scheduling

RESOURCE MANAGER
Overview

• The resource manager provides a persistent infrastructure of resources
  • Basically JVMs remotely accessible through a ProActive node

• It is responsible for
  • Acquiring (deploying) nodes from various sources
  • Maintaining a list of free/busy/dead nodes
  • Releasing nodes when they are not needed anymore
  • Providing nodes to clients

• A client can request an arbitrary number of resources
  • Asynchronous request

• 2 parts
  • RM server
  • RM client
Server - Client

- **RM server**
  - Selects nodes according to user requests
  - Restricts access to these nodes to other users
  - Monitors nodes states and provides up-to-date information about resources utilization

- **RM Client**
  - Request nodes from the server
  - Can add/remove nodes to the RM
  - Can trigger automatic deployment of new nodes
  - Example: Scheduler

- **Example of use**
  - A client contacts an existing RM server
  - Requests (and obtain) $n$ nodes
  - Performs a computation
  - Contacts the RM server to free the nodes when the computation is over
Node Source

• A Node Source is an object responsible for providing nodes to the RM
• Composed of an Infrastructure and a Policy
• Infrastructure
  • Responsible for actually deploying the nodes
  • Typically a frontend to an existing protocol/scheduler
• Policy
  • Defines rules/authorization on the node source
  • Who is the administrator, who are the users
  • When are the nodes actually deployed
Examples of Infrastructures

• **SSHInfrastructure**
  – Performs a SSH on a remote host and start a node

• **BatchJobInfrastructure**
  – Abstract Infrastructure for deploying on batching systems
  – Is specialized into *LSFInfrastructure, PBSInfrastructure, GEInfrastructure*…

• **EC2Infrastructure, WinHCPInfrastructure**
  – Amazon EC2, Windows HPC

• **VirtualInfrastructure**
  – Xen, VMWare, VirtualBox…
Example of Policies

• **StaticPolicy**
  - Deploy all nodes immediately

• **TimeSlotPolicy**
  - Deploy/release nodes at a specific time

• **SchedulerLoadingPolicy**
  - Deploy nodes when the Scheduler is overloaded
  - Listener based implementation

• **EC2Policy**
  - Extends **SchedulerLoadingPolicy**
  - Releases resources only in the last 10 minutes of the paid hour
  - Useful for Cloud Bursting
Authentication

• Use of standard *Java Authentication and Authorization Service*

• User/groups are stored
  • In a local (server side) file
  • A LDAP

• Local password file
  • Passwords in a **plain text** in a file of the RM server
  • Should only used for testing/debugging

• LDAP
  • Can connect to existing LDAP
Authorizations

- What can done with nodes
  - Who can get/release nodes
  - Who can create/remove nodes from a node source

- What can be done with node sources
  - Who can add/remove node sources
  - Limitations on nodes from a particular source to a set of users groups

- Authorizations are specified in a plain text file
  - $RM_HOME/config/security.java.policy-server.

- Limitation on node source are specified at creation time
  - Requires node source administration authorizations
Authorizations

grant principal org.ow2.proactive.authentication.principals.UserNamePrincipal "john" {
    permission org.ow2.proactive.permissions.MethodCallPermission
        "org.ow2.proactive.resourcemanager.core.RMCore.getAtMostNodes";
    permission org.ow2.proactive.permissions.MethodCallPermission
        "org.ow2.proactive.resourcemanager.core.RMCore.releaseNodes";
};

grant principal org.ow2.proactive.authentication.principals.GroupNamePrincipal "admin" {
    permission org.ow2.proactive.permissions.AllPermission;
};
Requesting nodes

- After authenticating with the RM server
- A client can request a number of nodes
  - Randomly
  - With some topological properties
  - With some arbitrary properties

```java
NodeSet nodeSet = resourceManager.getAtMostNodes(nbOfNodes, selectionScript);
```

- Call is asynchronous and returns a *future*
- Best effort only
Selection scripts

• How can we allow a user to select particular nodes
• Matchmaking?
  • Condor-like mechanism
  • Requires work from both administrators and users
  • Sometimes users don’t know what they need…
• Selection script
  • A script provided by the user to select nodes
  • If the script is executed successfully, the node is selected
• Pros:
  • Users do not need to learn a resource description language
  • Very flexible, ultimately script == small scale version of the real job
  • User can provides multiple scripts at once
• Cons:
  • Time consuming
  • Requires care (rogue scripts…)
Selection scripts

• How to reduce the cost of selecting nodes?
  • First: don’t execute the scripts! 😊
  • If you have to, then choose a machine which is very likely to pass the test

• Static vs. Dynamic scripts
  • Some characteristics are static (Total memory, CPU Frequency…)
  • Some are dynamics (Current free memory, current free disk space, available license token…)
  • Allow user to tag the script as static or dynamic
  • Static scripts are only run once on each machine

• Scoring of machines
  • Each couple node-script is given a selection probability
  • For each success (resp. failure) the probability increases (resp. decreases)
  • Test machines with highest probability first
Node Topology

- Select nodes based on topology
- **SingleHostDescriptor**: nodes on the same host
- **BestProximityDescriptor**: set of closest nodes (latency)
- **ThresholdProximityDescriptor**: set of nodes within a threshold proximity
- RM Server should be configured to discover node topology
Conclusion on resourcing

• The *Resource Manager* is very powerful
  • Can deploy any application, not just ProActive ones
  • It’s almost a PaaS

• But lacks support for easily starting tasks
  • Can theoretically fork JVMs, start virtual machines…
  • The user friendly classes are in the *Scheduler*

• However, it does not helps with firewalls, private networks…
  • Assume other solution solve the issue!
**Programming**

Java Parallel Toolkit
Accelerate & Scale your most demanding applications with Multi-Core and Distributed Parallel Computing:
- Portability, Fault-tolerance, File transfer, SOA, Security
- Master Worker, Monte Carlo, Legacy Wrapping, SPMD, File Split & Merge, Task Flow
- Asynchrony, groups, futures
- Monitoring & Optimizing GUI

**Resourcing**

Desktop, Cluster, Grid, Cloud Resource Manager
Leverage existing infrastructures:
- Private Cloud, Virtualization (VMware, KVM, Xen)
- Dynamic Resource Acquisition (including EC2, NVIDIA GPU)
- Deploy applications with ssh, rsh, sshGSI
- Schedule with PBS, LSF, SGE, Prun, EGEE gLite, IBM LL
- Windows and Linux Agents

**Scheduling**

Multi-Platform Job Scheduler
Schedule parallel jobs on all company resources and on external clouds:
- Fault-tolerance, Security
- File Transfer
- Java and XML Taskflows
- Java or native applications, scripts (Ruby, Python, JavaScript), Matlab, Scilab, MPI. ProActive Programming
Programming

MULTI-ACTIVE OBJECTS
Introduction

Shortcomings of the current model:

- No data sharing – slow local parallelism
- No re-entrant calls – changes to program logic

Goals:

- Transparent multi-threading
- Possibility to write re-entrant code
- Safe execution
- Ease of use
- Legacy support
Proposal

When are two methods compatible?
   If either:
     a) They do not access the same resources
     b) The user “protects” the places of possible data races

How to use this information?
   • Inside active objects, compatible requests can be served in parallel

How to express this?
   • Annotate the code
Proposal

Annotations

• Defining pair wise relation – too much work
• Groups
  – Collection of related methods
• Rules
  – Compatibility relationship between groups
• Membership
  – The group a method belongs to
Proposal – Annotations

@Group
- Group identifier
- Self compatible?
- Location: class header

@DefineGroups(
    {
        @Group(name="GroupF", selfCompatible=true),
        @Group(name="GroupB", selfCompatible=false)
    }
)
Proposal – Annotations

@Compatible
- A list of mutually compatible groups
- Location: class header

```java
@DefineRules({
    @Compatible( { "GroupF", "GroupB" } )
})
```
Proposal – Annotations

@memberOf

- Refers to a group identifier
- Location: before a method

```java
@memberOf("GroupF")
public int foo_1() {...}
@memberOf("GroupF")
public int foo_2() {...}
@memberOf("GroupB")
public int bar() {...}
```
Proposal

Scheduling

• Default: FIFO-like
• Possible to use policies (API)

```java
method runActivity() {
    while (true) {
        if (compatible(requestQueue.peekFirst(),
                        activeRequests)) {
            parallelServe(requestQueue.removeFirst());
        }
    }
}
```
Proposal

- Scheduling Policy API
  - Provides:
    - Static compatibility information
    - Scheduler state (request queue, active serves)
  - State does not change while evaluating a policy

- What is a policy?
  - A function
  - Input: state and compatibility
  - Output: a list of requests to be started
Scheduling with a policy

Annotated Java Source → Annotation Processor → Compatibility information → Queue → Scheduler → Policy → Multi-Active Object
Proposal – Code

Implementation

- org.objectweb.proactive.annotation.multiactivity
  - Annotations

- org.objectweb.proactive.multiactivity
  - Annotation processor
  - Interfaces and data types
  - Multi-Active Service
  - Policy Factory
Experiment

Distributed Graph Algorithm

- Strongly connected component search
- Divide-and-conquer – lots of parallelism

Given: \( G = (V, E) \).

procedure \( FB(V) \)
  pick pivot \( v \in V \);
  \( F := Fwd(v) \);
  \( B := Bwd(v) \);
  report \( F \cap B \);
  in parallel do
    \( FB(F \setminus B) \);
    \( FB(B \setminus F) \);
    \( FB(V \setminus (B \cup F)) \);
  end FB
Experiment

Sources of local parallelism
• Forward and backward marking is compatible
• Markings from different origins are compatible

Sources of global parallelism
• Graph is distributed among workers
• Markings propagate into neighbors
Experiment

Had to change the logic of the algorithm?
• No

Results
• Easy distributed execution – Grid5000
• Quite large graph sizes
• Acceptable speedup
Experiment – Results

Local parallelism

Performance
Future Work

A parallelism-maximizing policy

- Not considers only the top of the queue for parallel execution
- Ensures the same properties as we have now

Other “real-life” experiments on large clusters

- Still an open discussion 😊