ProActive Parallel Suite

Denis Caromel Arnaud Contes
Univ. Nice ActiveEon
Agenda

► ProActive and ProActive Parallel Suite
► Programming and Composing
  - ProActive Core
  - High Level Programming models
  - ProActive Components
► Deployment Framework
► Development Tools
Unification of Multi-Threading and Multi-Processing

**Multi-Threading**
Multi-Core Programming

- SMP
  - Symmetric Multi-Processing
  - Shared-Memory Parallelism

- Solutions: OpenMP, pThreads, Java Threads...

**Multi-Processing**
Distributed programming, Grid Computing

- MPP
  - Massively Parallel Programming or
  - Message Passing Parallelism

- Solutions: PVM, MPI, RMI, sockets, ...
ProActive

- ProActive is a JAVA middleware for parallel, distributed and multi-threaded computing.

- ProActive features:
  - A programming model
  - A comprehensive framework

To simplify the programming and execution of parallel applications within multi-core processors, distributed on Local Area Network (LAN), on clusters and data centers, on intranet and Internet Grids.
Unification of Multi-threading and Multi-processing

Seamless

Sequential  Multithreaded  Distributed

► Most of the time, activities and distribution are not known at the beginning, and change over time
► Seamless implies reuse, smooth and incremental transitions
ProActive Parallel Suite

ProActive Parallel Suite includes:

- The ProActive middleware featuring services like:
  - Fault tolerance, Load balancing, Distributed GC, Security, WS
  - A set of parallel programming frameworks
  - A framework for deploying applications on distributed infrastructures
- Software for scheduling applications and resource management
- Software for monitoring and profiling of distributed applications
- Online documentation
- Full set of demos and examples

2009
ProActive Parallel Suite

Applications

ProActive Parallel Suite

Physical Infrastructure
Ways of using Proactive Parallel Suite?

► To easily develop parallel/distributed applications from scratch

► Develop applications using well-known programming paradigms thanks to our high-level programming frameworks (master-worker, Branch&Bound, SPMD, Skeletons)

► To transform your sequential mono-threaded application into a multi-threaded one (with minimum modification of code) and distribute it over the infrastructure.
Ways of using Proactive Parallel Suite?

- To wrap your native application with ProActive in order to distribute it

- Define jobs containing your native-applications and use ProActive to schedule them on the infrastructure
Agenda

► ProActive and ProActive Parallel Suite
► Programming and Composing
  ❑ ProActive Core
  ❑ High Level Programming models
  ❑ ProActive Components
► Deployment Framework
► Development Tools
ProActive Parallel Suite

Developer Tools & Eclipse IDE Plugins
- IC2D Monitoring & Debugging
- Grid IDE
- Timit Profiling

Programming & Composing
- Master-Worker
- Monte-Carlo
- Event Programming
- MATLAB SCILAB
- Branch & Bound
- Skeletons
- Workflow
- SPMD

Core API
- GCM - Components
- Active Objects
- Asynchrony
- Futures
- Groups
- Mobile Agents
- MOP / AOP

Deployment & Virtualization
- GCM Deployment
- File Transfer
- Desktop P2P Grid
- Scheduler & Infrastructure Manager

Services
- Load Balancing
- Fault Tolerance
- Security
- Distributed Garbage Collector
- Web Services
ProActive Core

ACTIVE OBJECTS
ProActive

A 100% Java API + Tools for Parallel, Distributed Computing

▸ A programming model: Active Objects
  □ Asynchronous Communications, Wait-By-Necessity, Groups, Mobility, Components, Security, Fault-Tolerance

▸ A formal model behind: Determinism (POPL’04)
  □ Insensitive to application deployment

▸ A uniform Resource framework
  □ Resource Virtualization to simplify the programming
Active Objects

Developer writes Object A

new A(...) newActive(A,..)

With ProActive, he gets ...

Activity
Remote Accessible Objects
Asynchronous Communications
Code Mobility
Security
Fault Tolerance
Distributed Garbage Collector
Resource Virtualization

2009
ProActive model : Basis

► Active objects
  - coarse-grained structuring entities (subsystems)
  - has exactly one thread.
  - owns many passive objects (Standard Java Objects, no thread)
  - No shared passive objects -- Parameters are deep-copy

► Remote Method Invocation
  - Asynchronous Communication between active objects

► Full control to serve incoming requests

JVM
Active objects

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

\[ (a, \sigma) \rightarrow_{s} (a', \sigma') \]
\[ \alpha[a; \sigma'; \nu; F; R; f] \parallel P \rightarrow \alpha[a'; \sigma'; \nu; F; R; f] \parallel P \]  (LOCAL)

\[ \gamma \text{ fresh activity} \quad \nu' \notin \text{dom}(\sigma) \quad \sigma' = \{ \nu' \mapsto AO(\gamma) \} :: \sigma \]
\[ \sigma_{\gamma} = \text{copy}(\nu'', \sigma) \quad \text{Service} = ( \text{if } m_j = \emptyset \text{ then } \text{FifoService} \text{ else } \nu''.m_j()) \]
\[ \alpha[R[\text{Active}(\nu'', m_j)]; \sigma; \nu; F; R; f] \parallel P \rightarrow \alpha[R[\nu'''; \sigma'; \nu; F; R; f] \parallel \gamma[\text{Service}; \sigma_{\gamma}; \nu'''; \emptyset; \emptyset; \emptyset] \parallel P \]  (NEWACT)

\[ \sigma_{\alpha}(\nu) = AO(\beta) \quad \nu'' \notin \text{dom}(\sigma_{\beta}) \quad f_{i}^{\alpha-\beta} \text{ new future} \quad \nu_{f} \notin \text{dom}(\sigma_{\alpha}) \]
\[ \sigma'_{\beta} = \text{Copy} \& \text{Merge}(\sigma_{\alpha}, \nu''; \sigma_{\beta}, \nu'') \quad \sigma'_{\alpha} = \{ \nu_{f} \mapsto \text{fut}(f_{i}^{\alpha-\beta}) \} :: \sigma_{\alpha} \]
\[ \alpha[R[m_j(\nu'''); \sigma_{\alpha}; \nu_{f}; F_{\alpha}; R_{\alpha}; f_{\alpha}]; \beta[\sigma_{\beta}; \nu_{f}; F_{\beta}; R_{\beta}; f_{\beta}] \parallel P \rightarrow \alpha[R[\nu_{f}'; \sigma'_{\alpha}; \nu_{f}; F_{\alpha}; R_{\alpha}; f_{\alpha}]; \beta[\sigma_{\beta}; \nu_{f}'; \nu_{f}; F_{\beta}; R_{\beta}; \{ m_j; \nu'''; f_{i}^{\alpha-\beta}; f_{\beta} \}] \parallel P \]  (REQUEST)

\[ R = R' :: [m_j; \nu_{f}; f''] :: R'' \quad m_j \in M \quad \forall m \in M, m \notin R' \]
\[ \alpha[R[\text{Serve}(M)]; \sigma; \nu; F; R; f] \parallel P \rightarrow \alpha[m_j(\nu_{f''}); \text{fut}(\nu_{f''}); \emptyset; f, R[\emptyset]]; \sigma; \nu; F; R' :: R''; f''] \parallel P \]  (SERVE)

\[ \nu' \notin \text{dom}(\sigma) \quad F' = F \{ \nu \mapsto \nu' \} \quad \sigma' = \text{Copy} \& \text{Merge}(\sigma, \nu; \sigma', \nu') \]
\[ \alpha[\nu \uparrow (\nu', a); \sigma; \nu; F; R; f] \parallel P \rightarrow \alpha[a; \sigma'; \nu; F'; R; f'] \parallel P \]  (ENDSERVICE)

\[ \sigma_{\alpha}(\nu) = \text{fut}(f_{i}^{\alpha-\beta}) \quad F_{\beta}(f_{i}^{\alpha-\beta}) = \nu_{f} \quad \sigma'_{\alpha} = \text{Copy} \& \text{Merge}(\sigma_{\beta}, \nu_{f}; \sigma_{\alpha}, \nu) \]
\[ \alpha[a_{\alpha}; \sigma_{\alpha}; \nu_{f}; F_{\alpha}; R_{\alpha}; f_{\alpha}]; \beta[\sigma_{\beta}; \nu_{f}; F_{\beta}; R_{\beta}; f_{\beta}] \parallel P \rightarrow \alpha[a_{\alpha}; \sigma'_{\alpha}; \nu_{f}; F_{\alpha}; R_{\alpha}; f_{\alpha}]; \beta[\sigma_{\beta}; \nu_{f}; F_{\beta}; R_{\beta}; f_{\beta}] \parallel P \]  (REPLY)
ProActive Core

MIGRATION: MOBILE AGENTS
Mobile Agents: Migration

► The active object migrates with:
  ◐ its state
  ◐ all pending requests
  ◐ all its passive objects
  ◐ all its future objects

► Automatic update of references:
  ◐ requests (remote references remain valid)
  ◐ replies (its previous queries will be fulfilled)

► Migration is initiated by the active object itself

► Can be initiated from outside through any public method
Migration Strategies

► Forwarders
- Migration creates a chain of forwarders
- A forwarder is left at the old location to forward requests to the new location
- Tensioning: shortcut the forwarder chains by notifying the sender of the new location of the target (transparently)

► Location Server
- A server (or a set of servers) keeps track of the location of all active objects
- Migration updates the location on the server

► Mixed (Forwarders / Local Server)
- Limit the size of the chain up to a fixed size
ProActive Core

PROACTIVE 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

```java
A ag=(A)ProActiveGroup.newGroup("A", {{p1},...},{Nodes,...});
V v = ag.foo(param);
v.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
Two Representations Scheme

- **Typed group** 'A'
- **Group of objects** 'Group'

Management of the group

Functional use of the group

- `getGroup` method of class `Group`
- `getGroupByType` static method of class `ProActive`

2009
Creating AO and Groups

- A ag = newGroup ("A", [...], Node[])
- V v = ag.foo(param);
- ...
- JVM v.bar(); //Wait-by-necessity

![Diagram of object interactions]

- Typed Group
- Java or Active Object
Typed Group as Result of Group Communication

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

► Explicit Group Synchronization Primitive:
- Explicit wait
  - ProActiveGroup.waitOne(groupB);
  - ProActiveGroup.waitAll(groupB);
- Predicates
  - noneArrived
  - kArrived
  - allArrived, ...
ProActive Core

FAULT TOLERANCE SERVICE
Fault-tolerance in ProActive

► Restart an application from latest valid checkpoint
   □ Avoid cost of restarting from scratch

► Fault-tolerance is non intrusive
   □ set in a deployment descriptor file
   □ Fault-tolerance service attached to resources
   □ No source code alteration
      ▪ Protocol selection, Server(s) location, Checkpoint period
Fault-tolerance in ProActive

► Rollback-Recovery fault-tolerance
  - After a failure, revert the system state back to some earlier and correct version
  - Based on periodical checkpoints of the active objects
  - Stored on a stable server

► Two protocols are implemented
  - Communication Induced Checkpointing (CIC)
    + Lower failure free overhead
    - Slower recovery
  - Pessimistic Message Logging (PML)
    - Higher failure free overhead
    + Faster recovery

► Transparent and non intrusive
Built-in Fault-tolerance Server

- Fault-tolerance is based on a global server
- This server is provided by the library, with
  - Checkpoint storage
  - Failure detection
    - Detects fail-stop failures
  - Localization service
    - Returns the new location of a failed object
  - Resource management service
    - Manages a set of nodes on which restart failed objects
ProActive Core

SECURITY SERVICE
ProActive Security Framework

Issue
Access control, communication privacy and integrity

► Unique features
  □ SPKI: Hierarchy of certificates
  □ No security related code in the application source code
  □ Declarative security language
  □ Security at user- and administrator-level
  □ Security context dynamic propagation

► Configured within deployment descriptors
  □ Easy to adapt according the actual deployment
ProActive Core

WEB SERVICES
Web Service Integration

► Aim
- Turn active objects and components interfaces into Web Services
  - interoperability with any foreign language or any foreign technology.

► API
- Expose an active object as a web Service (the user can choose the methods he wants to expose)
  - `exposeAsWebService(Object o, String url, String urn, String [] methods );`
- Expose component’s interfaces as web services
  - `exposeComponentAsWebService(Component component, String url, String componentName );`
1. ProActive.exposeAsWebService()  
2. Deployment  
3. Client Call  

ProActive.exposeAsWebService(…………)

-WSDL file Urn= ‘piComputation’

3. Client Call

- .NET
- C#
Agenda

► ProActive and ProActive Parallel Suite
► Programming and Composing
  ❑ ProActive Core
  ❑ High Level Programming models
  ❑ ProActive Components
► Deployment Framework
► Development Tools
High Level Programming models

Master-Worker Framework
Motivations

► Embarrassingly parallel problems: simple and frequent model

► Write embarrassingly parallel applications with ProActive:

- May require a sensible amount of code (fault-tolerance, load-balancing, ...).

- Requires understanding of ProActive concepts (Futures, Stubs, Group Communication)
Goals of the M/W API

► Provide a easy-to use framework for solving embarrassingly parallel problems:
  - Simple Task definition
  - Simple API interface (few methods)
  - Simple & efficient solution gathering mechanism

► Provide automatic fault-tolerance and load-balancing mechanism

► Hide ProActive concepts from the user
How does it work?
Comparison between specific implementation and M/W

- Experiments with nQueens problem
- Runs up to 25 nodes
High Level Programming models

Skeletons Framework
Algorithmic Skeletons

- High Level Programming Model
- Hides the complexity of parallel/distributed programming.
- Exploits nestable parallelism patterns

Task Parallelism

Data Parallelism

- farm
- while
- divide & conquer

- pipe
- for
- map
- if
- fork
Skeletons Big Picture

- Parameters/Results are passed through streams
- Streams are used to connect skeletons (CODE)

Skeleton Code

Parameter (Data) → Input Stream → Skeleton Code → Output Stream → Solved: Results
Pipe Skeleton

- Represents computation by stages.
- Stages are computed in parallel for different parameters.

Input Stream → Execute Skeleton → Output Stream
Simple use of Pipe skeleton

```java
Skeleton<Eggs, Mix> stage1 =
    new Seq<Eggs, Mix>(new Apprentice());

Skeleton<Mix, Omelette> stage2 =
    new Seq<Mix, Omelette>(new Chef());

Skeleton<Eggs, Omelette> kitchen =
    new Pipe<Eggs, Omelette>(stage1, stage2);
```
High Level Programming models

Branch-and-Bound Framework
Branch & Bound API (BnB)

► Provide a high level programming model for solving BnB problems:
  - manages task distribution and provides task communications

► Features:
  - Dynamic task split
  - Automatic result gather
  - Broadcasting best current result
  - Automatic backup (configurable)
Global Architecture : M/W + Full connectivity
High Level Programming models

OO-SPMD
Object-Oriented Single Program Multiple Data

- **Motivation**
  - Cluster / GRID computing
  - SPMD programming for many numerical simulations
  - Use enterprise technology (Java, Eclipse, etc.) for Parallel Computing

- **Able to express most of MPI’s**
  - Collective Communications (broadcast, gathercast, scattercast,..)
  - Barriers
  - Topologies

- **With a small object-oriented API**
A ag = newSPMDGroup ("A", [...], VirtualNode)
    // In each member
    myGroup.barrier ("2D"); // Global Barrier
    myGroup.barrier ("vertical"); // Any Barrier
    myGroup.barrier ("north","south","east","west");
Topologies

- Topologies are typed groups
- Customizable
- Define neighborhood

```java
Plan plan = new Plan(groupA, Dimensions);
Line line = plan.getLine(0);
```
High Level Programming models

Scheduler
Programming with flows of tasks

► Program an application as an ordered tasks set
  - Logical flow: Tasks execution are orchestrated
  - Data flow: Results are forwarded from ancestor tasks to their children as parameter

The task is the smallest execution unit

Two types of tasks:
  - Standard Java
  - Native, i.e. any third party application
Defining and running jobs with ProActive

▶ A workflow application is a job
  - a set of tasks which can be executed according to a dependency tree

▶ Rely on ProActive Scheduler only

▶ Java or XML interface
  - Dynamic job creation in Java
  - Static description in XML

▶ Task failures are handled by the ProActive Scheduler
  - A task can be automatically re-started or not (with a user-defined bound)
  - Dependant tasks can be aborted or not
  - The finished job contains the cause exceptions as results if any
Agenda

► ProActive and ProActive Parallel Suite
► **Programming and Composing**
  - ProActive Core
  - High Level Programming models
  - ProActive Components
► Deployment Framework
► Development Tools
ProActive Parallel Suite

Developer Tools & Eclipse IDE Plugins
- IC2D Monitoring & Debugging
- Grid IDE
- Timit Profiling

Services
- Load Balancing
- Fault Tolerance
- Security
- Distributed Garbage Collector
- Web Services

Programming & Composing
- Master-Worker
- Monte-Carlo
- Event Programming
- MATLAB SCILAB
- Branch&Bound
- Skeletons
- Workflow
- SPMD

Core API
- Active Objects
- Asynchrony
- Futures
- Groups
- Mobile Agents
- MOP / AOP

Deployment & Virtualization
- GCM Deployment
- File Transfer
- Desktop P2P Grid
- Scheduler & Infrastructure Manager
A framework for Grid components

► Facilitating the design and implementation of complex distributed systems

► Leveraging the ProActive library
  ProActive components benefit from underlying features

► Allowing reuse of legacy components (e.g. MPI)

► Providing tools for defining, assembling and monitoring distributed components
Component - What is it?

► A component in a given infrastructure is:

- a software module,
- with a standardized description of what it needs and provides,
- to be manipulated by tools for Composition and Deployment.
ProActive Component Definition

► A component is:
  - Formed from one (or several) Active Object
  - Executing on one (or several) JVM
  - Provides a set of server ports: Java Interfaces
  - Uses a set of client ports: Java Attributes
  - Point-to-point or Group communication between components

► Hierarchical:
  - Primitive component: define with Java code and a descriptor
  - Composite component: composition of primitive + composite
  - Parallel component: multicast of calls in composites

► Descriptor:
  - XML definition of primitive and composite (ADL)
  - Virtual nodes capture the deployment capacities and needs

► Virtual Node:
  - a very important abstraction for GRID components
Components for the GRID

1. Primitive component
   - An activity, a process, ... potentially in its own JVM

2. Composite component

Composite: Hierarchical, and Distributed over machines

Parallel: Composite + Broadcast (group)

3. Parallel and composite component
Components vs. Activity and JVMs

- Components are orthogonal to activities and JVMs
  - They contain activities, span across several JVMs

- Components are a way to globally manipulate distributed, and running activities
Agenda

► ProActive and ProActive Parallel Suite
► Programming and Composing
  ❑ ProActive Core
  ❑ High Level Programming models
  ❑ ProActive Components
► Deployment Framework
► Development Tools
GCM Deployment

Developer Tools & Eclipse IDE Plugins
- IC2D Monitoring & Debugging
- Grid IDE
- Timit Profiling

Services
- Load Balancing
- Fault Tolerance
- Security
- Distributed Garbage Collector
- Web Services

Programming & Composing
- High-Level Programming Models & Legacy Code Wrapping

Core API
- Active Objects
- Asynchrony
- Futures
- Groups
- Mobile Agents
- MOP / AOP

GCM - Components

Deployment & Virtualization
- GCM Deployment
- File Transfer
- Desktop P2P Grid
- Scheduler & Infrastructure Manager

2009
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
Resource Virtualization

Application

VN1

VN2

GCM XML Deployment Descriptor

Host

JVM

node

node

node

2009
Virtualization resources

VN1

VN2

Application

Host

JVM

node

node

JVM

node

JVM

node
Multiple Deployments

- One Host
- Local Grid
- Distributed Grids

Internet
Rmissh : SSH Tunneling

► A fact : overprotected clusters
  - Firewalls prevent incoming connections
  - Use of private addresses
  - NAT, IP Address filtering, …

► A consequence :
  - Multi clustering is a nightmare

► Context :
  - SSH protocol : encrypt network traffic
  - Administrators accept to open SSH port
  - SSH provides encryption
Rmissh : SSH Tunneling (2)

- Create a communication protocol within ProActive that allows firewall transversal

- Encapsulates rmi streams within ssh tunnels

- Avoid ssh tunneling costs when possible by first trying a direct rmi connection then fallbacking with rmissh
The ProActive P2P
The ProActive P2P

- Unstructured P2P
  - Easier to deploy/manage
  - Only 1 resource: CPU

- Java code
  - Each peer is written in Java and can run any Java application

- Direct communications
  - Peers are reachable using their name (URLs)
  - One peer can send/receive a reference on another peer
The ProActive P2P (2)

Applications

Resource Management

Direct Access

P2P Infrastructure
Infrastructure

- A peer is an Active Object in a JVM
- Each peer knows a limited number of other peers (bi-directional links)
  - Its acquaintances
  - The number is set by a variable (NOA)
- Goal of a peer
  - A peer will always try to maintain the number of its acquaintances equals to its NOA
- 2 basic operations
  - Adding an acquaintance
  - Removing an acquaintance
Requesting Nodes

- To request a node
  - Contact only a Peer (URLs)

- The infrastructure will handle the reservation

- The application has to wait until the nodes are available

- Using the P2P network
  - Programmatically at runtime using the Java API
  - At Deployment time through the GCMDeployment
Scheduler and Resource manager
Scheduler / Resource Manager Overview

- Multi-platform Graphical Client (RCP)
- File-based or LDAP authentication
- Static Workflow Job Scheduling, Native and Java tasks, Retry on Error, Priority Policy, ConfigurationScripts,…
- Dynamic and Static node sources, Resource Selection by script, Monitoring and Control GUI,…
- ProActive Deployment capabilities: Desktops, Clusters, ProActive P2P,…

2009
Agenda

- ProActive and ProActive Parallel Suite
- Programming and Composing
  - ProActive Core
  - High Level Programming models
  - ProActive Components
  - Legacy code wrapping
- Deployment Framework
- Development Tools
IC2D
Interactive Control & Debug for Distribution

▸ Basic Features:
  □ Graphical visualization
  □ Textual visualization
  □ Monitoring and Control

▸ Extensible through RCP plug-ins
  □ TimIt
  □ ChartIt
  □ P2P view
  □ DGC view
IC2D: Monitor your application in real-time
TimIt: Automatic Timers in IC3D
Analysis and Optimization
M/W Success Story: Artificial Life Generation

Sylvain Cussat-Blanc, Yves Duthen – IRIT TOULOUSE

**Application**

Development of artificial creatures

**ProActive Version**

<table>
<thead>
<tr>
<th>Initial Application (C++)</th>
<th>1 PC</th>
<th>56h52 =&gt; Crashed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProActive Version</td>
<td>300 CPUs</td>
<td>19 minutes</td>
</tr>
</tbody>
</table>
Price-It workload distribution with ProActive

- Low level parallelism: shared memory
- Written in c++
- Originally written for Microsoft compiler
- JNI, Com interface
- No thread safe

- Upgrading the code base to thread safe code might be costly
- Is there any easier and cheaper alternative to extract parallelism from Price-it Library?
CPS : C++ API Client for ProActive Scheduler

► CPS : Client for ProActive Scheduler
► Shipped as .so/.dll
► A set of C++ methods to submit jobs to the Scheduler
  ❑ SchedulerClient::init() and dispose()
  ❑ SchedulerClient::submitJob(Job* jobPtr)
  ❑ SchedulerClient::getJobResult(int jobId)
► Internally uses JNI
Using CPS in Price-It

JVM spawned by CPS

Price-It (Master) C++

NCPS C++ / Java JNI bridge

Classical Java Scheduler Client

ProActive Scheduler Java

Price-It (Worker) C++ .dll

Price-It (Worker) C++ .dll

Price-It (Worker) C++ .dll

Workers are shipped as .dll then loaded by JVMS and executed through JNI
Conclusion

► Simple and Uniform programming model
► Rich High Level API
► Write once, deploy everywhere (GCMD)

► Let the developer concentrate on his code, not on the code distribution

► Easy to install, test, validate on any network
Now, let’s play with ProActive…

- **Start** and **monitor** with IC2D the ProActive examples, and have a look at the **source code** org.objectweb.proactive.examples.*

<table>
<thead>
<tr>
<th>Features</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basics, Synchronization</td>
<td>Doctors problem (doctors.bat), Reader/Writer problem (readers.bat),...</td>
</tr>
<tr>
<td>Futures, Automatic Continuation</td>
<td>Binary Search Tree (bintree.bat)</td>
</tr>
<tr>
<td>Migration</td>
<td>Migrating Agent (/migration/penguin.bat)</td>
</tr>
<tr>
<td>Group</td>
<td>Chat (/group/chat.bat)</td>
</tr>
<tr>
<td>Fault-Tolerance</td>
<td>N-body problem (/FT/nbodyFT.bat)</td>
</tr>
<tr>
<td>All</td>
<td>Distributed 3D renderer (c3d*.bat)</td>
</tr>
</tbody>
</table>