Java Parallel Proccessing Framework

An Open Source alternative to grid computing

http://www.jppf.org/

Agenda

- What is JPPF?
- Features at a glance
- JPPF Architecture
- Low cost deployment
- Simple APIs
- Robustness and scalability
- Security
- Administration and monitoring
- Roadmap
- Q & A

What is JPPF?

- General-Purpose Grid Toolkit
  - Federate computing resources working together
  - Handle large computational applications
  - Handle data-intensive problems
- A Java Framework
  - Ubiquitous programming platform
  - OS and hardware independent
  - A platform for integration, extension, customization
- An Open-Source Grid Environment
  - Flexible licensing (LGPL)
  - Source code guarantees transparency
  - Community-driven development process

Features at a glance

- Low-cost deployment
  - Rapid installation and setup in existing networks
  - Deploy to one location, execute anywhere on the grid
  - Minimal cost of updates and maintenance
  - Easily adapt to infastructure changes
- Highly scalable
  - Small networks: dozens of nodes
  - Large networks: thousands of nodes
  - Very large networks: millions of nodes
- Built-in failover and recovery
  - Embedded in every component of the framework
  - Flexible topology enables a high level of redundancy

Features at a glance

- Simple, efficient APIs
  - Focus on the application, not the grid framework
  - Object-oriented
  - Minimally intrusive on existing or legacy code
  - Small footprint
- Multiple deployment options
  - Standalone Java nodes
  - OS services
  - Oportunistic grid: JPPF@Home
  - Efficiently adapts to Corporate and Scientific infrastructure needs

Features at a glance

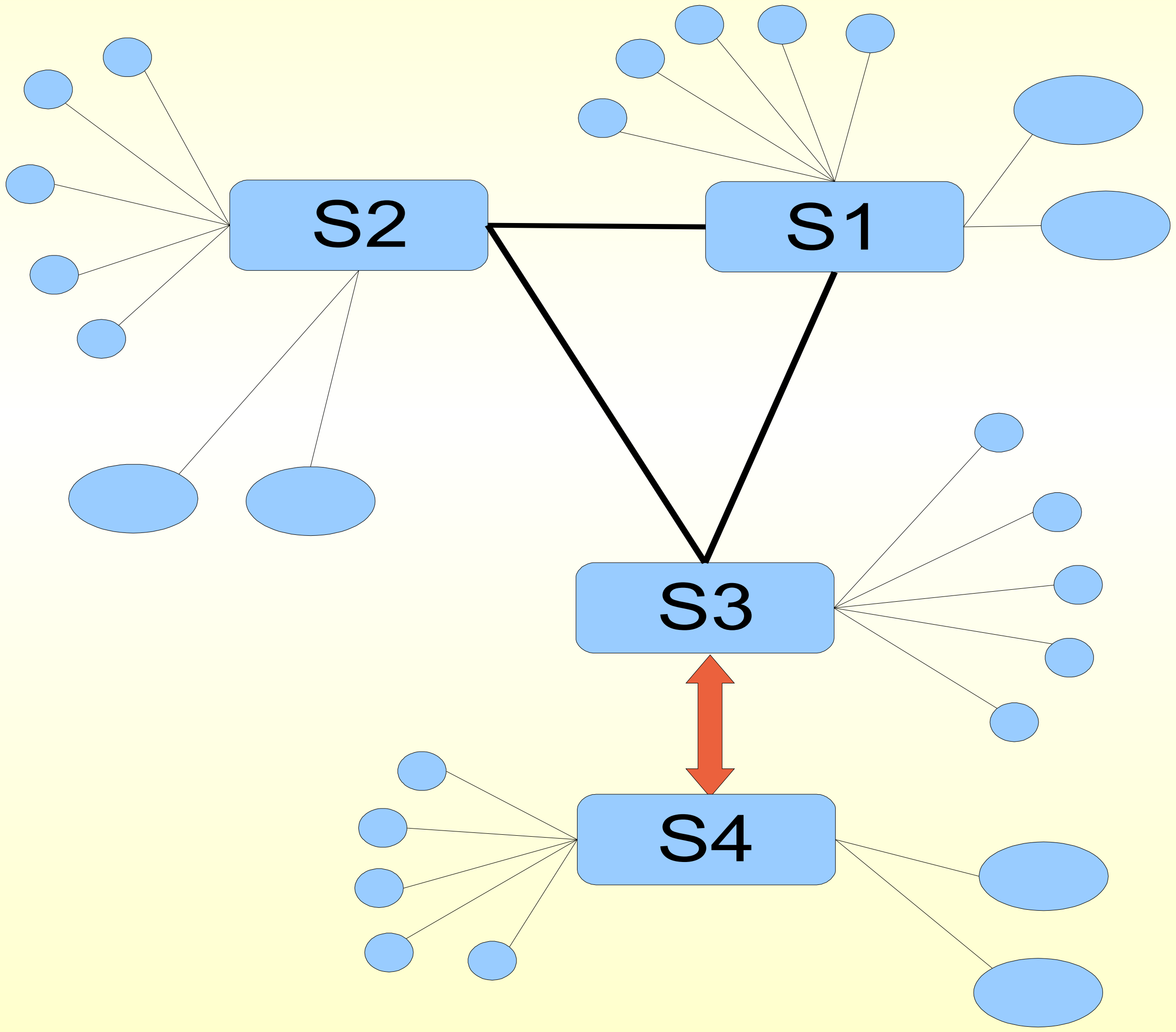
- A secure framework
  - Finely configurable security level
  - Secure interactions between distributed components
  - Integration in existing security infrastructure
- Management and monitoring
  - Extensible management console
  - JMX-based management
  - Charting and statistics logging
  - Localized User Interface
- Flexibility of integration
  - With other grid toolkits
  - With other business and scientific solutions

JPPF Architecture

- Scalable distributed communication model
  - Consistent protocol between components
  - Adaptive load balancing
  - Optimized bandwith usage
- Robustness
  - Built-in failover
  - Finely tunable recovery behavior
  - No single point of failure
- High performance
  - Small framework overhead
  - Asynchonous, non-blocking I/O
  - Continuous, feedback driven performance optimization

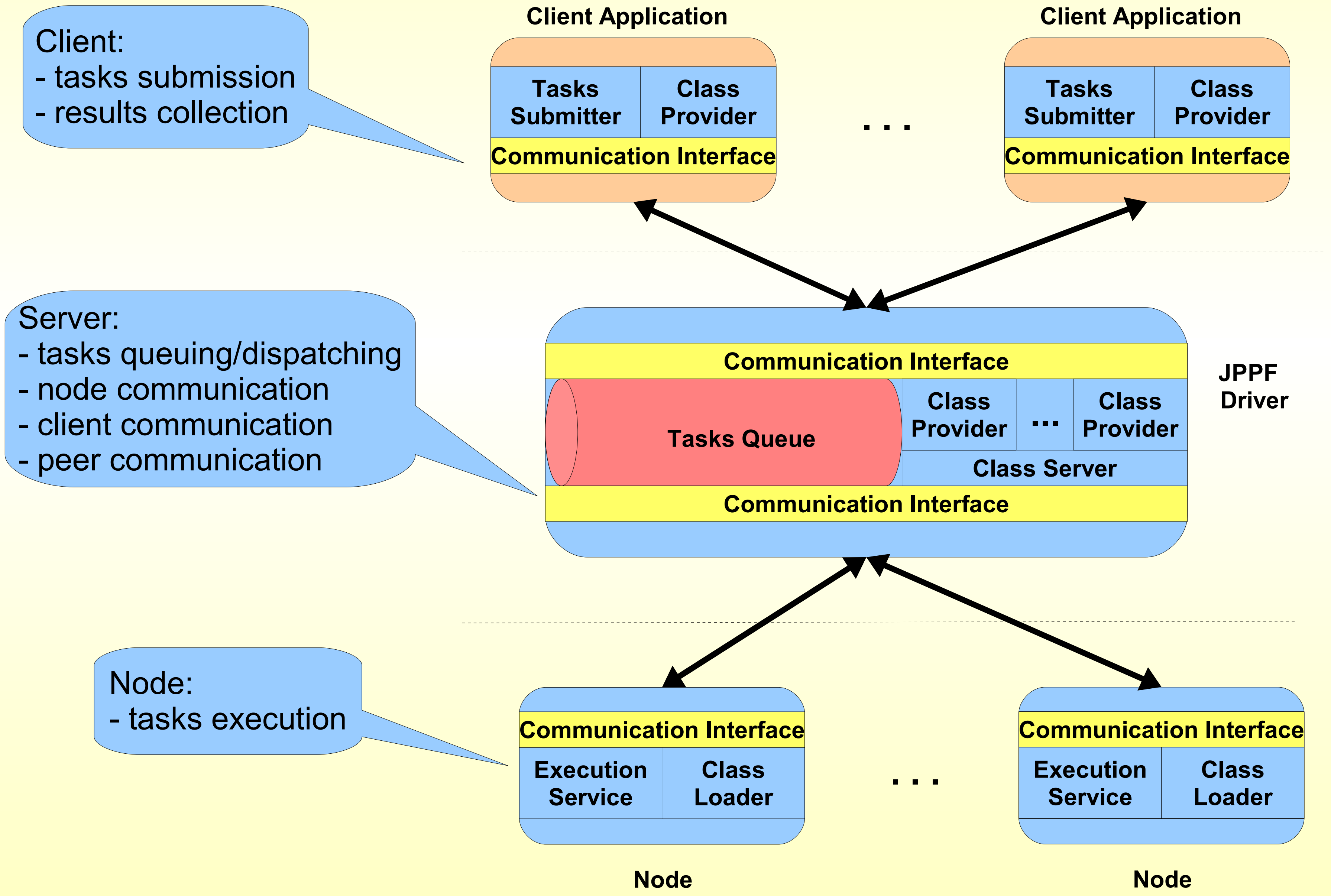
JPPF Architecture

Peer-to-peer, redundant topology



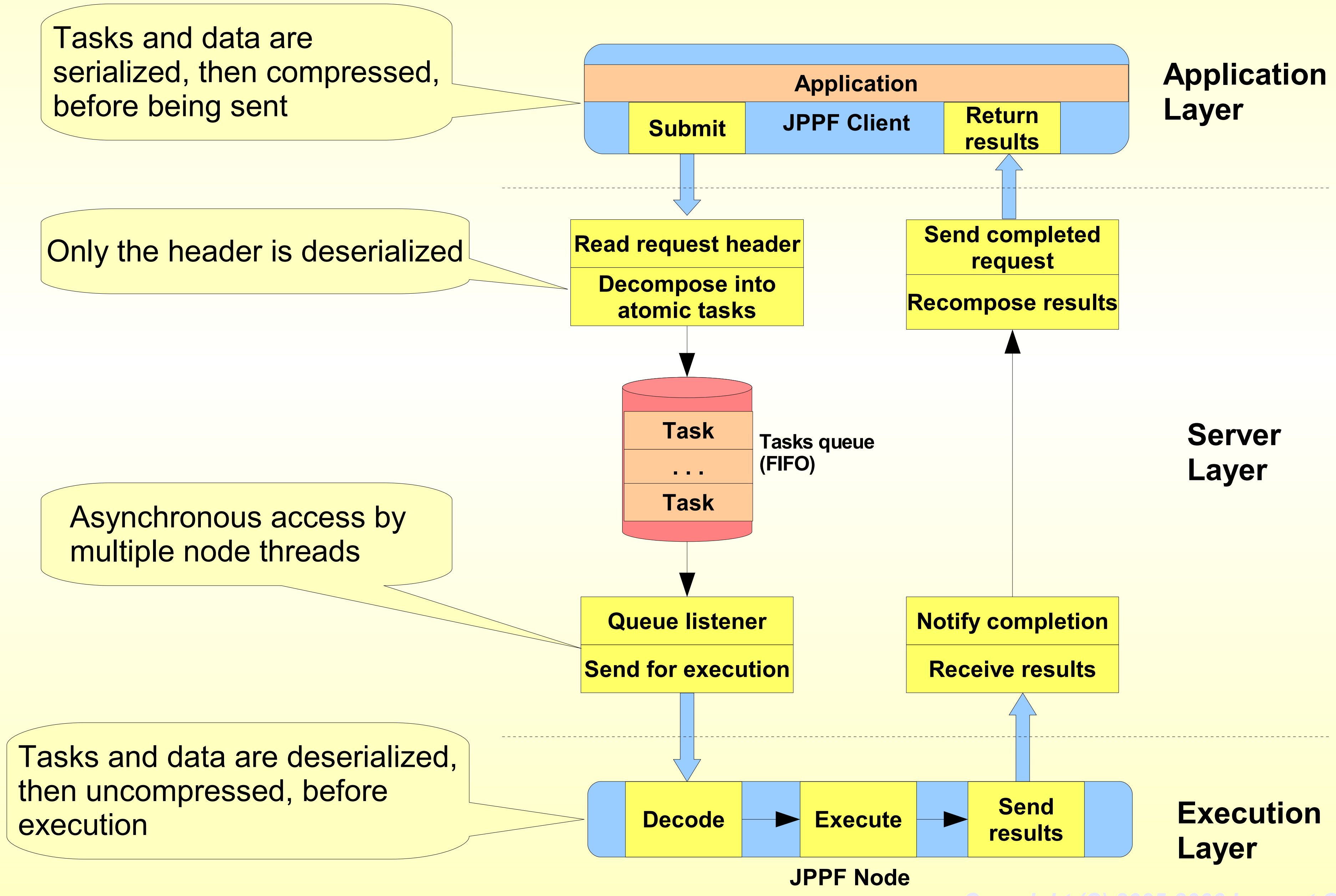
JPPF Architecture

Clear, efficient engineering



JPPF Architecture

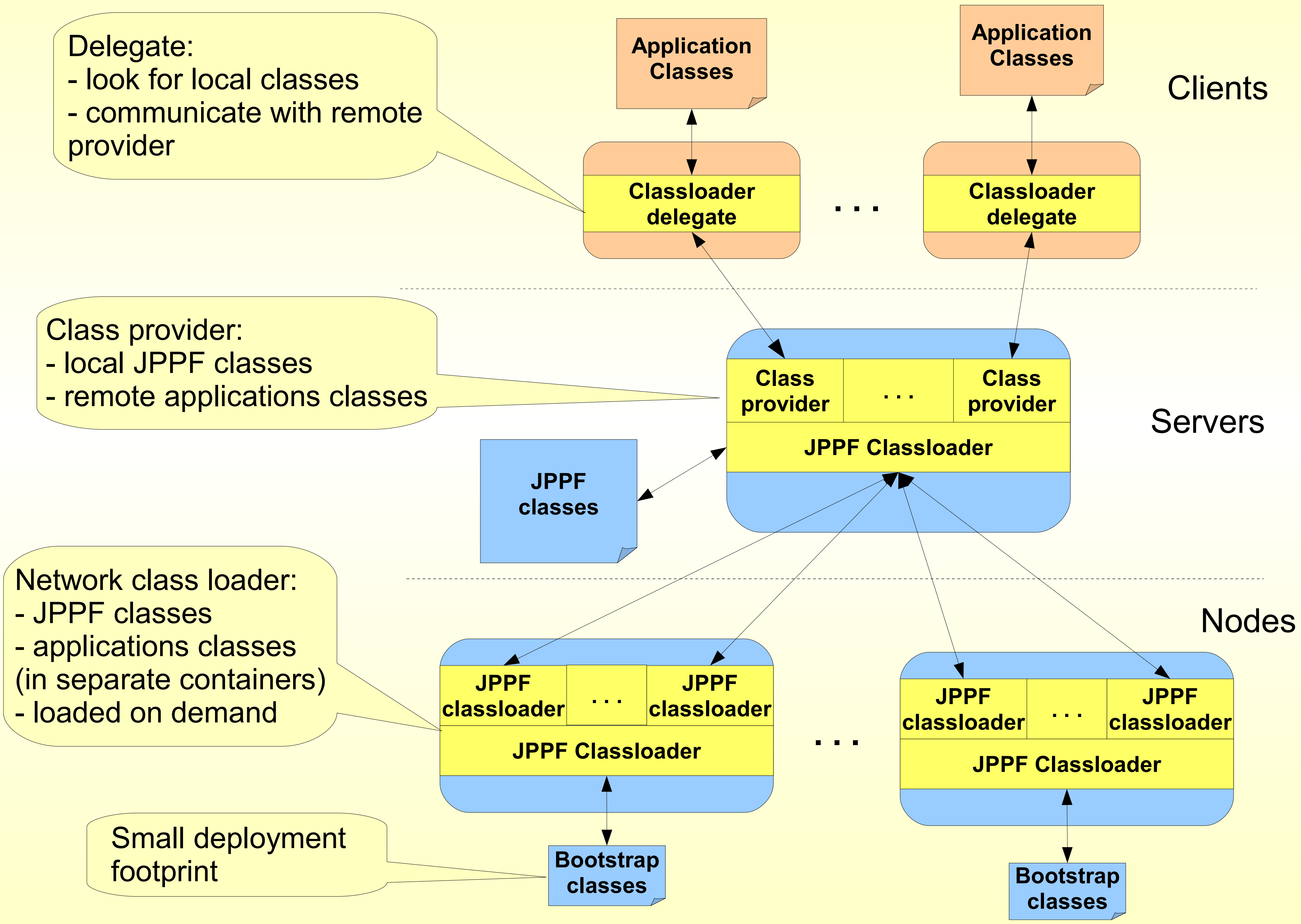
Optimized execution flow



Low cost deployment 1/2

- Code libraries deployed in one location
  - Transparently loaded on-demand by the nodes
  - Distributed loading over the entire grid
  - No classpath configuration
- Applications and JPPF automated updates
  - new/updated libraries automatically detected and loaded
  - Multiple versions can co-exist without risk
- Extension of Java technology
  - Based upon Java class loading features
  - Extension to a networked, highly distributed mechanism
  - Transparent to application developers

Low cost deployment 2/2



Simple APIs

A glimpse of the API: defining an atomic task

```java
/**
 * This class performs my parallelized code.
 * @author JPPF Developer
 */
public class MyTask extends JPPFTask
{
  /**
   * This is where it happens.
   * @see java.lang.Runnable#run()
   */
  public void run()
  {
    // here it begins
    ... my code ...
    // here it ends
    setResult(theComputationResult);
  }
}
```

Simple APIs

A glimpse of the API: submitting tasks for execution

```
/**
 * This class submits atomic tasks for execution.
 */
public class TaskSubmitter
{
  /**
   * Submit the tasks and wait for their execution.
   * @throws Exception if any exception is raised.
   */
  public void submit() throws Exception
  {
    JPPFClient jppfClient =  new JPPFClient();
    List<JPPFTask> taskList = new ArrayList<JPPFTask>();
    taskList.add(new MyTask(x));
    taskList.add(new MyTask(y));
    jppfClient.submit(taskList, null);
    System.out.println("The result of the first task is: "
      + taskList.get(0).getResult());
  }
}
```

Robustness and scalability

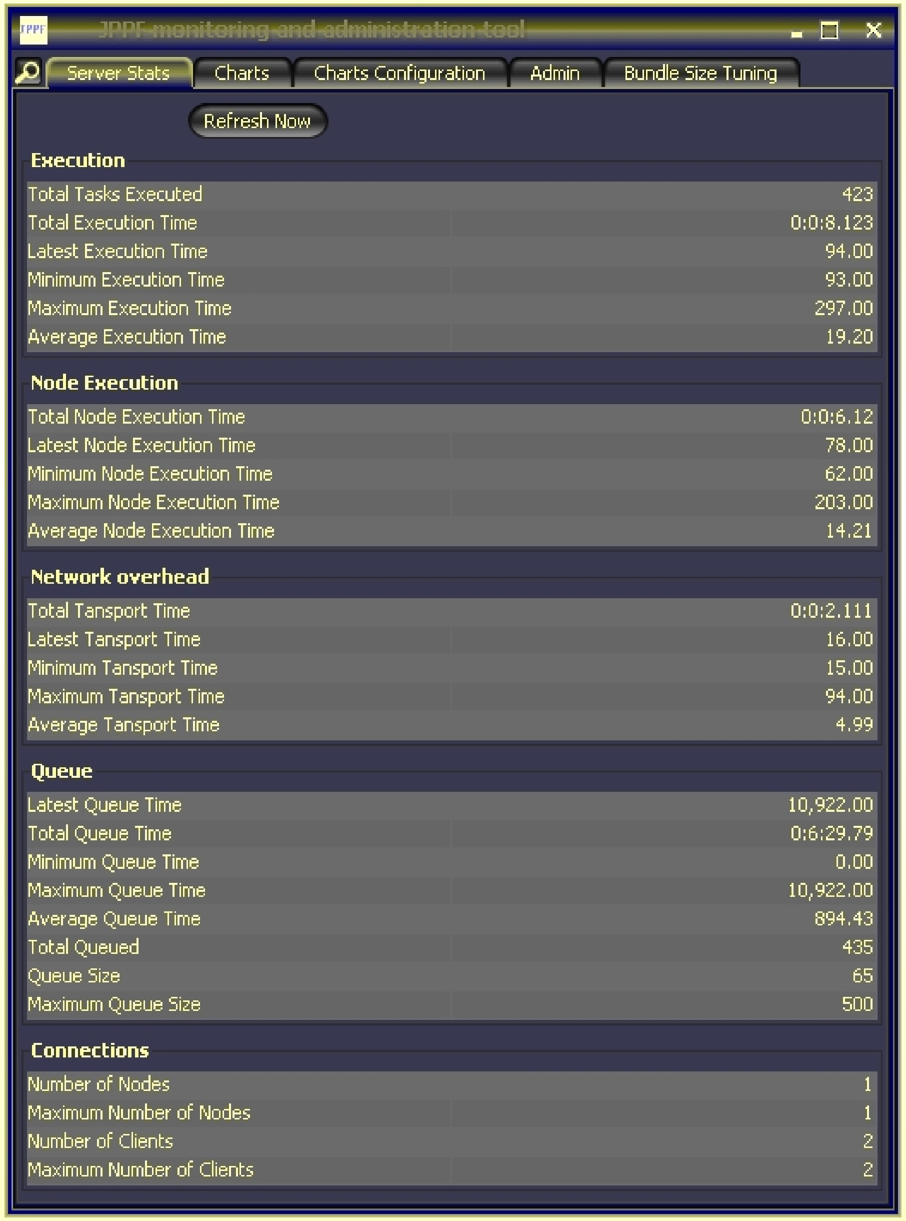
- Adaptive grid topology
  - Dynamically shrinks or grows with available resources
  - Adaptive load balancing ensures efficient routing
  - Enables collaboration between heteregenous networks
  - Works with local, private or public networks of any size
- Failover and recovery
  - Nodes and clients reconnect automatically
  - Clients and servers automatically resubmit tasks
  - Dynamically configurable to adapt to the environment
  - Servers can be restarted without losing attached resources
  - Redundancy ensures graceful degraded mode

Security

- Secure nodes
  - File system access restrictions
  - Restricted network connectivity
  - Restricted access to system environment
  - Access rights customizable through policy configuration
- Code confidentiality is guaranteed
  - Runs in an isolated container
  - Servers don't see it as code
- Secure cooperative work
  - JPPF components exchange credentials and authorizations
  - Easily integrates in existing security infrastructure

Administration and monitoring 1/3

Statistics dashboard



Execution

Total Tasks Executed	423
Total Execution Time	0:0:8.123
Latest Execution Time	94.00
Minimum Execution Time	93.00
Maximum Execution Time	297.00
Average Execution Time	19.20

Node Execution

Total Node Execution Time	0:0:6.12
Latest Node Execution Time	78.00
Minimum Node Execution Time	62.00
Maximum Node Execution Time	203.00
Average Node Execution Time	14.21

Network overhead

Total Tansport Time	0:0:2.111
Latest Tansport Time	16.00
Minimum Tansport Time	15.00
Maximum Tansport Time	94.00
Average Tansport Time	4.99

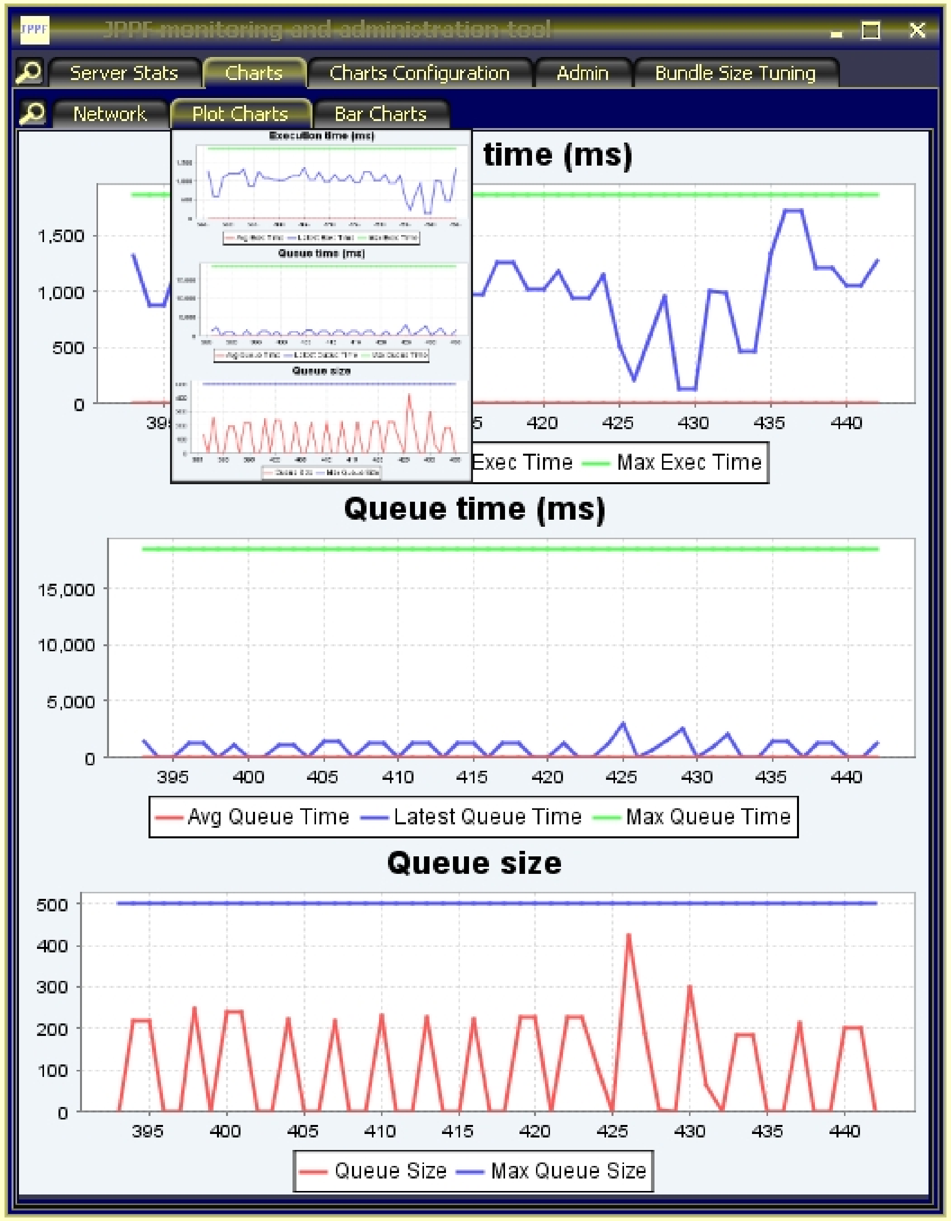
Queue

Latest Queue Time	10,922.00
Total Queue Time	0:6:29.79
Minimum Queue Time	0.00
Maximum Queue Time	10,922.00
Average Queue Time	894.43
Total Queued	435
Queue Size	65
Maximum Queue Size	500

Connections

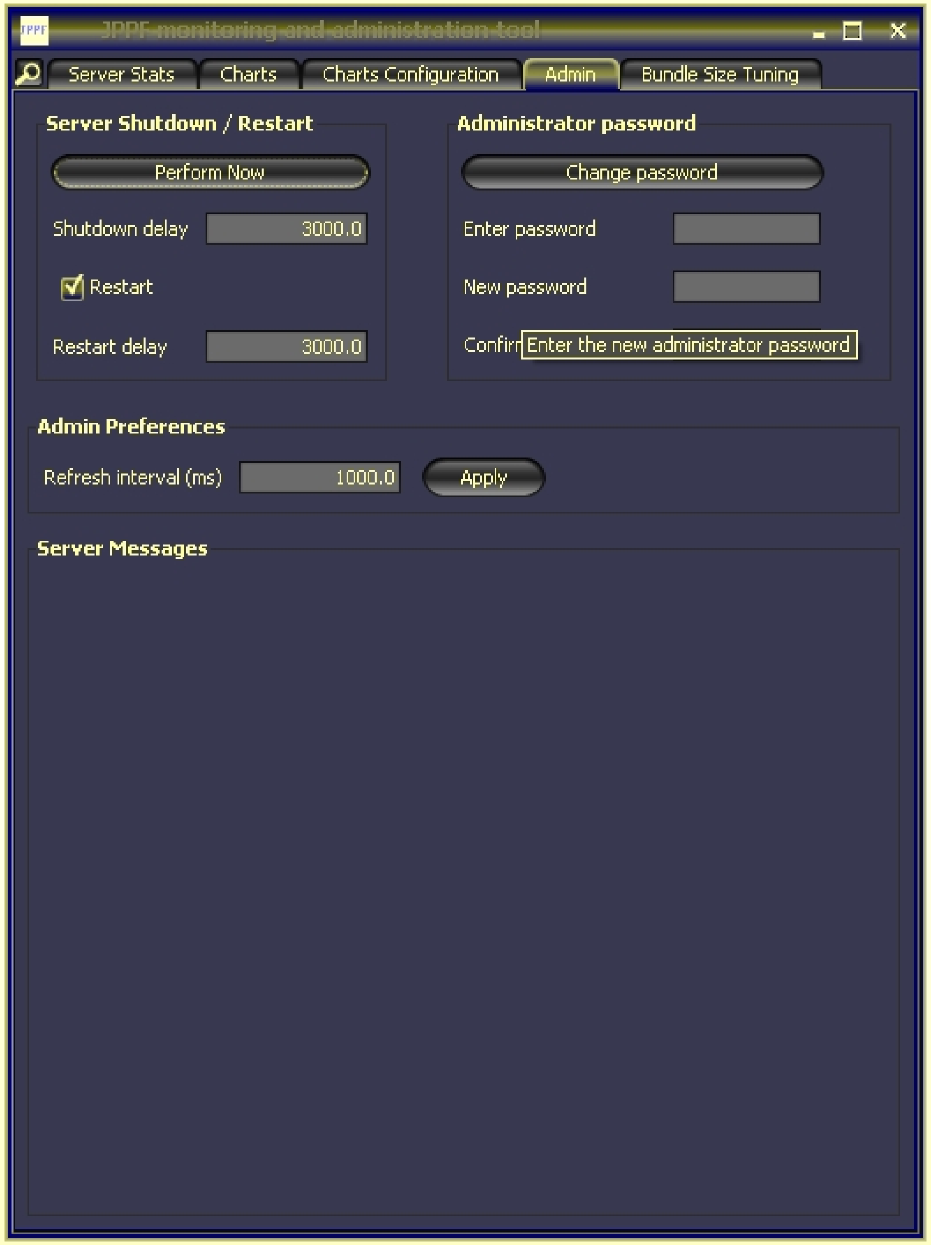
Number of Nodes	1
Maximum Number of Nodes	1
Number of Clients	2
Maximum Number of Clients	2

Full charting capabilities



Administration and monitoring 2/3

Remote administration



Server Shutdown / Restart

Shutdown delay	3000.0
✓ Restart	
Restart delay	3000.0

Administrator password

Enter password	
New password	
Confirm	

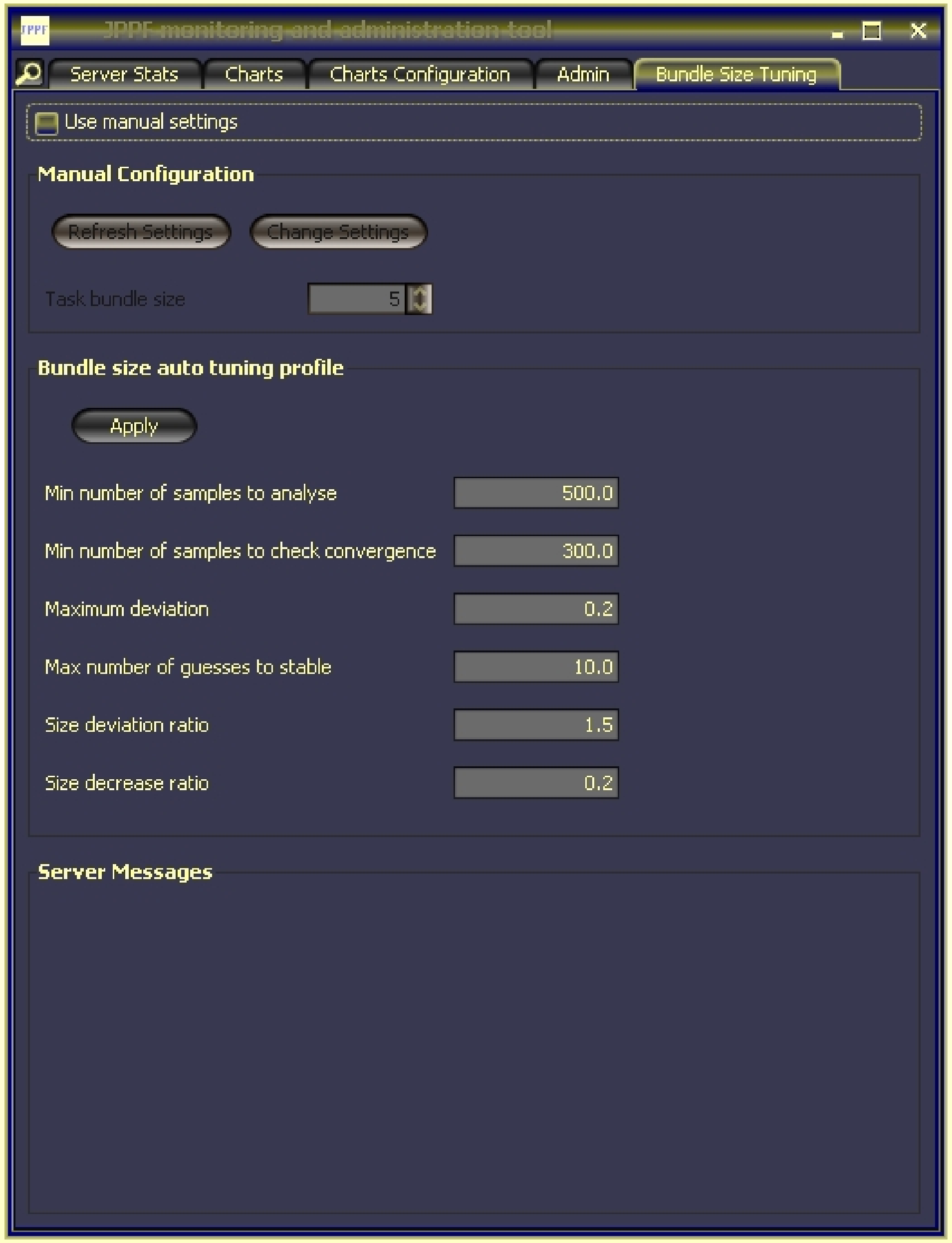
Enter the new administrator password

Admin Preferences

Refresh interval (ms)	1000.0

Server Messages

Remote tuning



☐ Use manual settings

Manual Configuration

Task bundle size	5

Bundle size auto tuning profile

Min number of samples to analyse	500.0
Min number of samples to check convergence	300.0
Maximum deviation	0.2
Max number of guesses to stable	10.0
Size deviation ratio	1.5
Size decrease ratio	0.2

Server Messages

Administration and monitoring 3/3

- Fully customizable graphical interface
  - Create your own charts
  - Dynamically generated from XML descriptors
  - Internationalization support
  - GUI builder tool
- Scriptable interface
  - Embedded scripting APIs
  - Ready for Java 1.6 scripting
  - Javascript reference implementation (Rhino)

Roadmap

Project vision

- ETL integration
- Business Intelligence back-end solution
- Globus toolkit integration
- J2EE and Web Services integration
- JMX-based management and monitoring
- Framework management automation
- Plugable services

Q & A

Thank You