



Load balancing in SOAJA (Service Oriented Java Adaptive Applications)

Richard Olejnik

Université des Sciences et Technologies de Lille

Laboratoire d'Informatique Fondamentale de Lille (LIFL UMR CNRS 8022)

Richard.Olejnik@lifl.fr

<http://www.lifl.fr/PALOMA/ADAJ>

Projet INRIA DOLPHIN



Overview

- ✓ **Introduction**
- ✓ **SOAJA environment**
- ✓ **Initial object deployment**
- ✓ **Dynamic object placement management**
- ✓ **Conclusion & Perspectives**

Motivations

- **Efficient load balancing on Grid platform**
- **Distribution models:** static, dynamic, adaptive
- **Distribution management**
 - load metrics for java computing
 - strategies
 - Information: distributed, centralized
 - Decision: threshold, selective transfers
 - mechanisms
 - Migration
 - Agent support, others



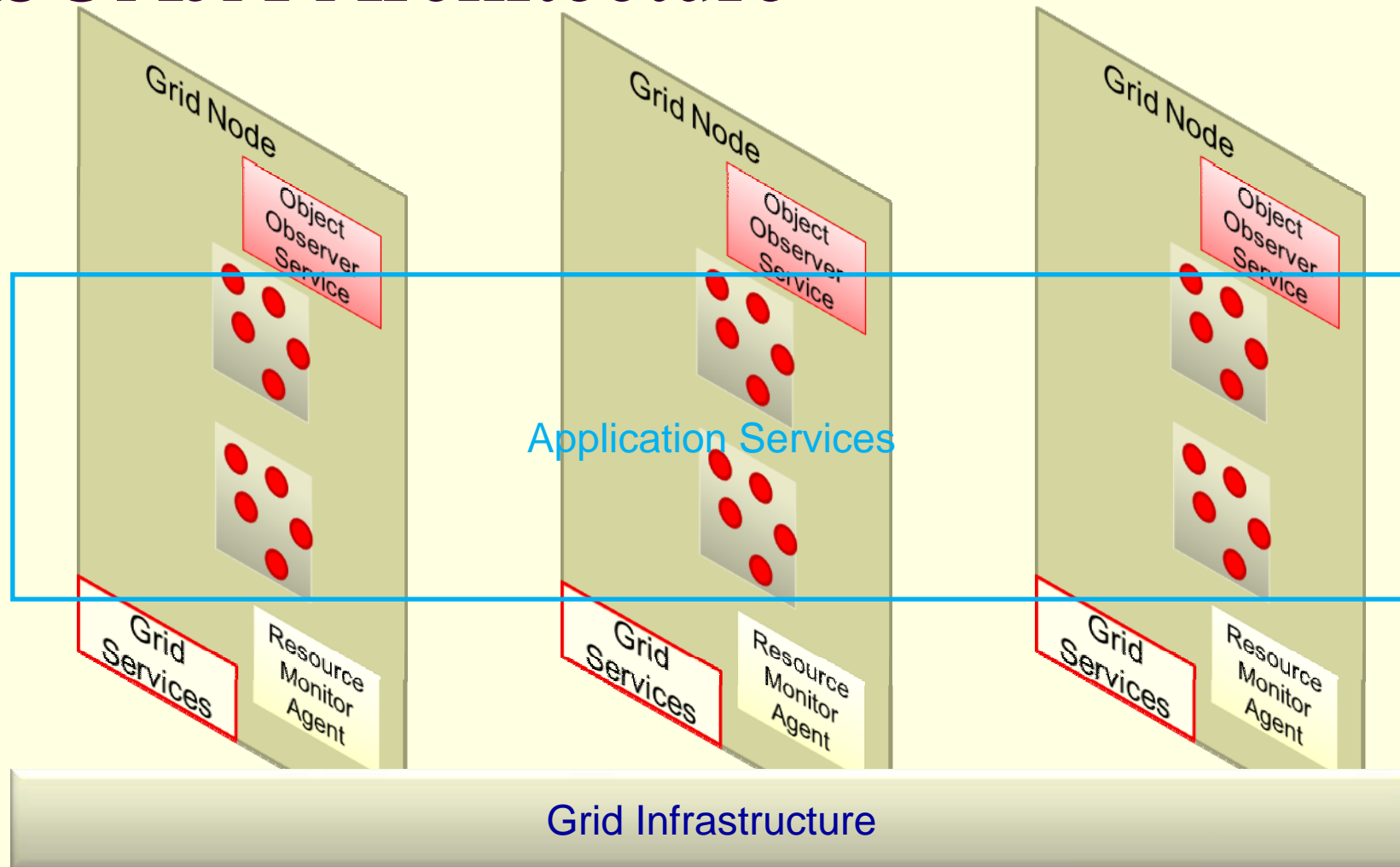
SOAJA environment characteristics

- SOA based
- Activities are implemented by means of *services*
- Integration with other SOA frameworks and the Grid
 - OGSA, OGSF
 - Grid services are expressed in WSDL
- Interoperability
- Self containment of services
- Loosely-coupled services and applications

Principles of SOAJA

- **Parallelism control and management transparency**
 - Facilitate the programmer work
 - Ensure effective implementation of parallelism
 - Inter and intra-application load balancing
- **Library of tools for parallel programming**
- **Observation system**
 - Scans the environment during execution
 - Retrieve information necessary to optimize the program
- **Load of the JVM and the physical machine**
 - Based on the information gathered by the observation
 - Detection of load imbalance
- **Correction of load imbalance**
 - Migration of objects from over-loaded to under-loaded machines

SOAJA Architecture





Integration of the environment

- Services level
 - Application services
 - Service Oriented Application (ex: DataMining)
 - Grid services
 - WSRF, WS-Management, Security, Information
 - SOAJA Services
 - Observation
 - Load Balancing
- Distributed Programming layer
 - Underling communication of *Objects*
 - Java objects communications

Application Services

- Deployment of Services
 - Description of the nodes
 - Placement of service objects
 - Finally all is objects
- Communication between objects
 - Java communication
 - Remote method Invocation
 - Distribution layer
 - Transparency to the service level

SOAJA Services

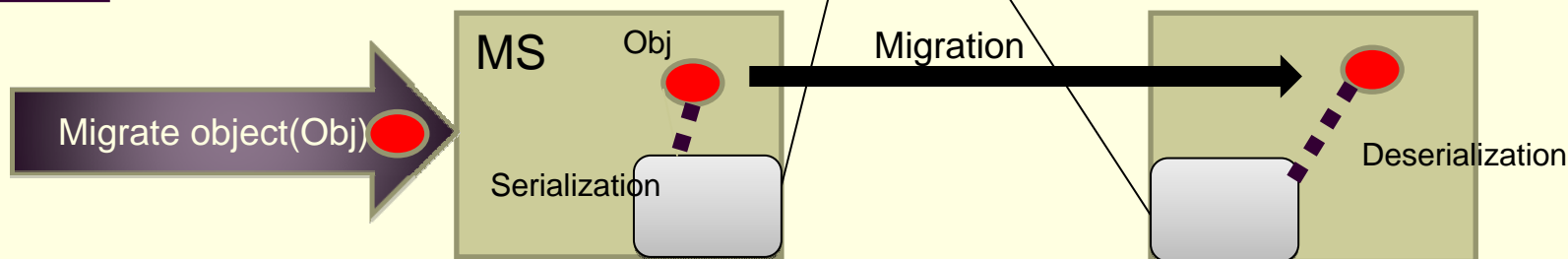
- Observer Services
 - Deployed on each node
 - Communicate with the agent system
 - Provide information about the objects in the node
- Load Balancing Services
 - Load computing
 - Determine under-loaded and over-loaded machines
 - Migrate objects

Example: Migration Service (MS)

- Service on each node
- Decision of the migration
 - Migration demand on the MS
 - Creation of serialization and deserialization of objects

ThreadMigrationSource

ServiceMigrationDestination

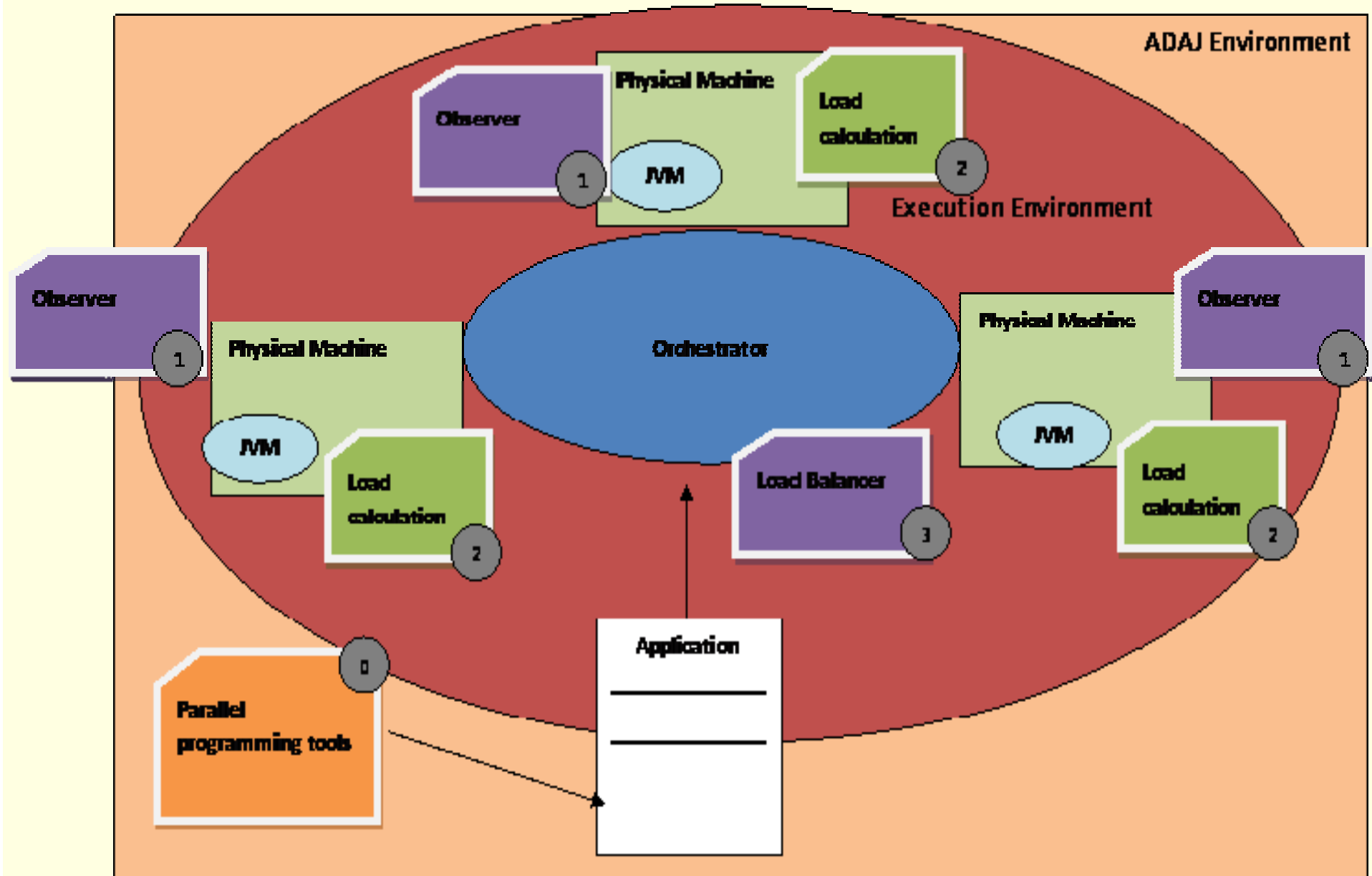




SOAJA Services Tools

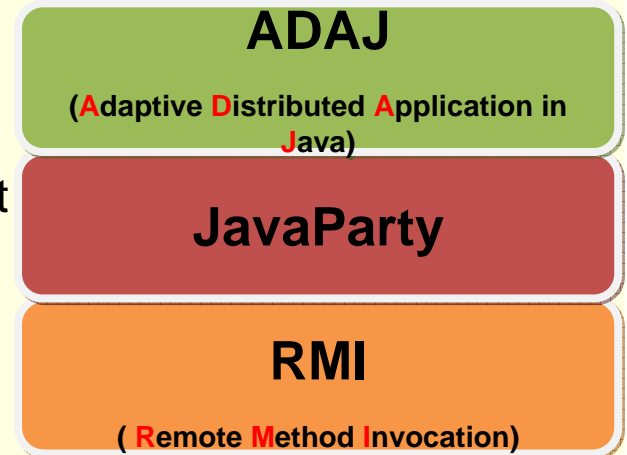
- Parallel Tools
 - In term of services
 - Facilitate parallelism control
 - Gateway to the underling environment
- Orchestrator
 - Management of service execution
 - ESB based
 - Initial deployment of services

SOAJA execution architecture



Distribution Layer

- Earlier ADAJ
 - Heavily based on JavaParty
 - Some incompatibilities
 - Special static environment management
 - Distributed Shared Memory
 - + transparency
 - + object migration facilities
 - pre-compiler
 - centralized controller (not scalable)
- New ADAJ = SOAJA
 - Services
 - Scalability to the Grid
 - Java based underling communications (without JavaParty)





Underlying SOAJA Environment

- Object Layer
 - Initial placement of Objects
 - Object Monitoring policy
 - Types of Objects
- Methods invocations
 - Monitoring method calls
- Object migration

Initial object deployment

- The most a node is overloaded the less it receives object to compute
- Function of the load on the JVM
 - dependant of the number of threads and of the quantity of the JVM work
 - in accord with the decision policy
- Initial object deployment based on graph analysis

Dynamic object management

- **Object monitoring**
 - gives the intensity of communication between objects
 - determines what objects could be migrated
- **Load monitoring**
 - predicts workstation load and network utilization
 - determines when perform the load balancing
 - **principle:** the average idle thread time is directly related to the CPU load

Object monitoring

- **SOAJA objects:**
 - **Global objects:**
 - remote creation
 - remote access
 - migratable
 - **Local Objects:**
 - traditional Java objects
 - copy creation
- **Only global objects are observed**

Observation mechanisms

- **Concerned objects:** global objects = remote objects
- **Properties**
 - Dynamic management
 - distributed objects' graph
- **Observed items**
 - quantity of objects' work
 - intensity of communications between objects
 - dynamics in time (smoothing of the values)
- **Java portability** (post-compilation)

Object monitoring strategies

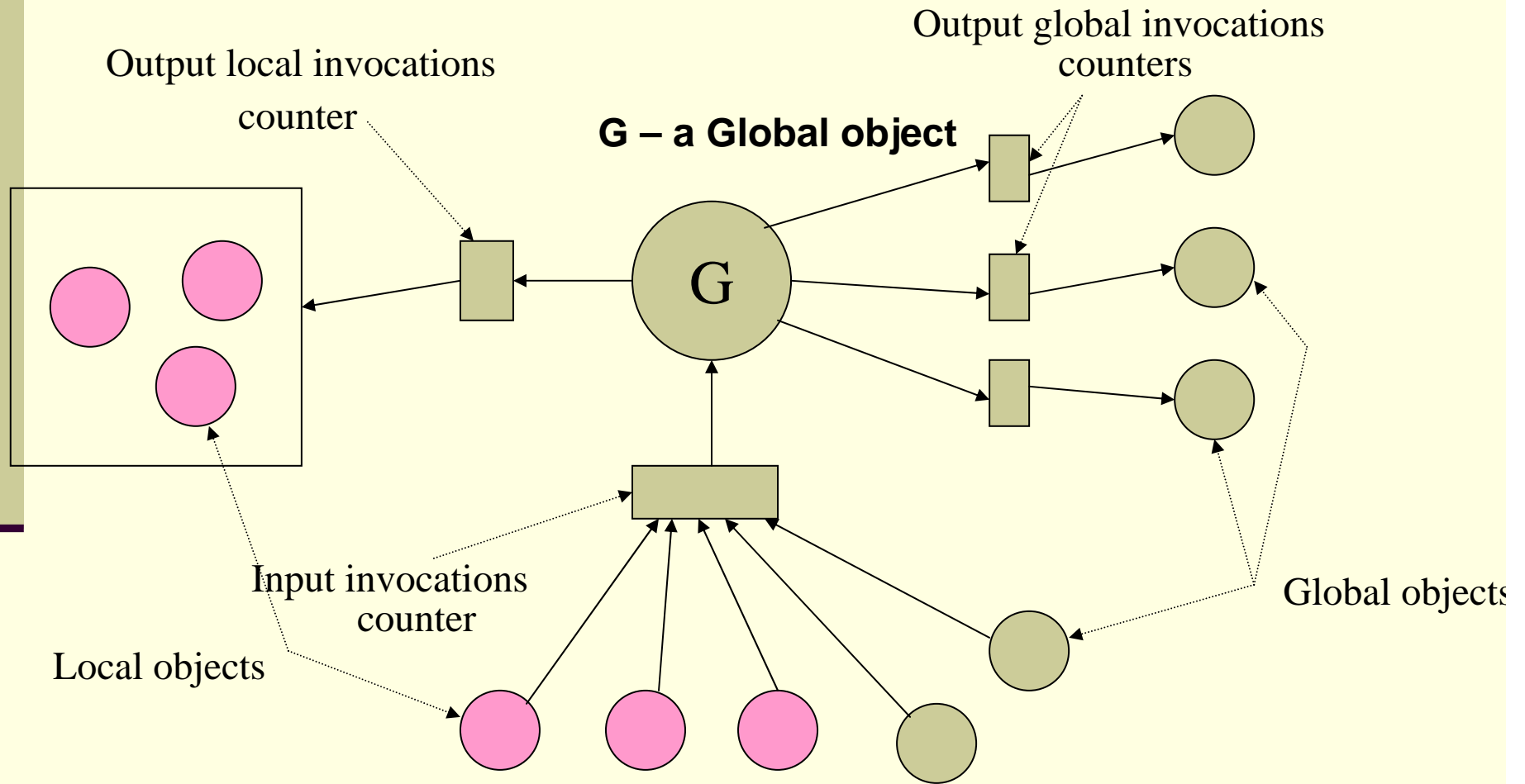
- **Object activity monitoring :**
 - estimation of the method running time
 - depends on the power of the computer node
- **Process communication monitoring :**
 - Measuring the exchanged information quantity
 - Computed based on the serialized code
 - 80% to 90% of the object transfer time is due to serialization
- **Counting the method invocation number (remote communication)**

Counting method invocations

- **Principle:** counting the invocations of a global object o_i
 - to each other global object o_j → remote communication
 - to all local objects → local communication
 - to itself (input invocations) → local work

- **Implementation:** For each global object o_i :
 - an array or other structures which memorizes the invocation number to other global objects
 - one counter for local object invocations
 - one counter for input invocations

Invocation counters



Observation mechanisms of the workstation load

- **Estimation of the load of one workstation**
 - load due to the application
 - external load
 - weighting relative to the workstation power
- **Principle**
 - computing of the average CPU idle time
- **Load balancing**
 - Information management system based on agents
- **Portability: use of Java threads**

Object redistribution

- **Load observation policy**
 - detection of abnormal situations
 - manage by agents (centralized, P2P)
 - load metrics
- **Policy for the migration candidate selection**
 - distributed
 - criteria choice
 - agregation
- **Target node selection policy**
- **Object transfer policy**

Load modeling in SOAJA

- Number of threads
- Ready threads and blocked threads (waiting)
- Quantity of a JVM work
 - sum of work quantity of each global objet
 - the work quantity of one global objet is generated by :
 - All outgoing calls to global objects
 - all incoming calls (II)
 - all outgoing calls toward local objects (OLI)

$$WP_{obj} = OGI(obj, obj) + II(obj) + OLI(obj)$$

Load observation policy

- Detection of load imbalance among nodes
- Computing each average node load
 - statistical measurements
 - mean, standard deviation
 - variation coefficient
 - K-Means algorithm
- Independent thresholds for the application and the measured values

Migration candidate selection policy

■ Principle

- classification of the global objects
- choice of the best candidate

■ Classification criteria

- attraction of global object to the current JVM

- $$attr(obj) = \sum_{o \in JVM} (OGI(obj, o) + OGI(o, obj))$$

- number of calls towards other global objects

■ Weight of a global objet

- quantity of work

$$Classification(obj) = \alpha_{attr} * \%attr(obj) + (1 - \alpha_{attr}) * \%dist_{mwp}(obj)$$

■ Combining the criteria

Target node selection policy

Principle

- classification of potential target JVM
- choice of the best target

Criteria for the JVM classification

- object attraction towards the target node
- work quantity of the target node

Combining the criteria

$$attrent_i = \sum_{object \in JVM_i} (OGI(object, obj) + OGI(obj, object))$$

Object mobility in SOAJA

■ Migration of global objects

- strong migration of an active method: data and state of the stack
- forced mobility

■ Implementation

- insertion of migration points (techniques of post-compilation)
- backup of the stack in a specific data structure: serialization techniques
- restart of the stopped method: techniques of exceptions

Conclusion

- SOAJA platform
- Initial optimized object deployment
- Execution efficiency: a mechanism of dynamic load balancing
 - metrics for the load balancing
 - attraction functions
 - aggregation of criteria
- Observation of the relations between objects of one application & exploitation of a distributed graph objects
- Centralized or distributed detection, distributed correction of the imbalance (agent system)



Perspectives

- Proof of the running correctness of such system
- Formal proof of the mechanisms of such adaptive load balancing
- Modeling multiple application running (starvation problem, ping pong effects ...)