Job scheduling in agent-based Grid middleware

Mehrdad Senobari, *Michał Drozdowicz*, Maria Ganzha, Marcin Paprzycki, Ivan Lirkov, Richard Olejnik, Pavel Telegin

Agenda

- Agents and the Grid
- *Agents in Grid* system
  - Architecture
  - Assumptions
- Existing grid schedulers
  - Traditional
  - Agent based
- Which approach best fits our system
- Conclusion

The need for agents in Grid

- Global Grid → collection of heterogeneous nodes belonging to “anyone”
  - Nodes can appear and disappear
  - Their load can change
  - The amount of nodes is large
  - QoS and SLA difficult to enforce
- ”...current Grid systems are somewhat rigid and inflexible in terms of their interoperation and their interactions, while agent systems are typically not engineered as serious distributed systems that need to scale, that are robust, and that are secure...”

Ian Foster, Nicholas R. Jennings, Carl Kesselman

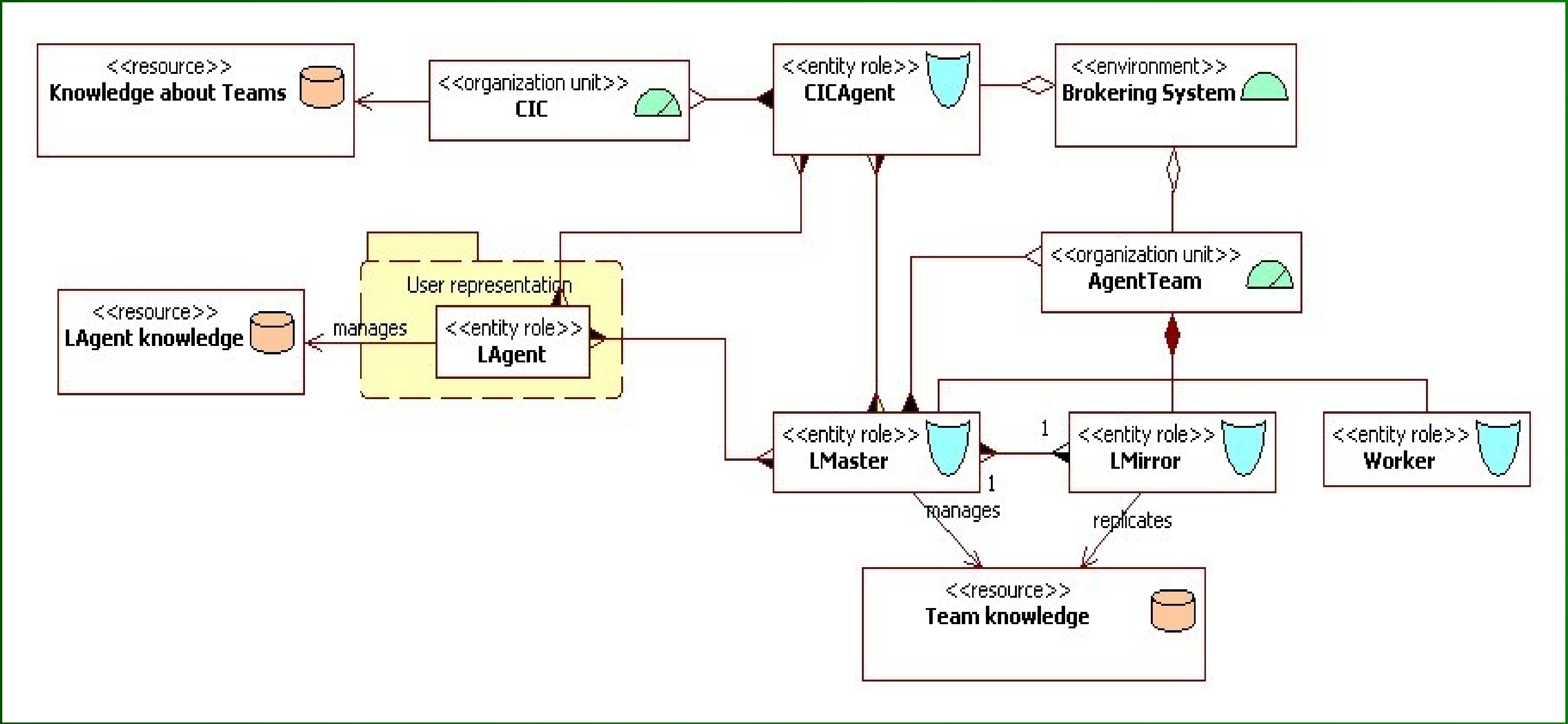
Software Agents as Resource Brokers in Grid

Goal: to enable Users to sell or/and buy resource(s)

Structure:

- Every resource is represented by an agent
- Agents works in teams
- *LMaster* – team manager; *LMirror* – "back-up man"
  - Share knowledge about team, jobs, clients...
- Advertising / Discovery / Registration → *Client Information Center* (*CIC*) infrastructure (*CIC Agent*)
  - All teams are registered in the *CIC DB*
  - *User* agent *LAgent* has to register in the *CIC*

AML social diagram of the system



Two main scenarios

- *User* needs (a) resource(s) to complete a task:

  1) Describes to *LAgent* requirements

  2) *LAgent* sends a request to the *CIC* → receives an answer (list of appropriate teams Managers – *LMasters*)

  3) *LAgent* negotiates conditions of task completion

  4) Returns results of the completed task to its *User*

- *User* wants to sell “resource(s)” (e.g. CPU time)

  1)-2) the same

  3) *LAgent* negotiates with *LMaster* conditions of work

  4) *LAgent* “works for a team” (earns money for its *User*)

Assumptions

- Known from job contract negotiations
  - requested hardware / software
  - execution constraints (including time and budget)
- Known from team joining
  - hardware / software of each Worker
  - time-focused contract details
- Agents are benevolent → does not mean omnipotent → reliability still important
  - malevolent agents (spoilers) have to be dealt with through “trust management” → very interesting (but for now out of scope)

CONDOR

- High-throughput environment
  - can manage a large collection of diversely owned machines
  - utilizes a centralized scheduler based on the ClassAd matchmaker, which manages a pool of available computing resources
  - allows the matchmaker (and/or the user) to forward computing request to another matchmaker through the gateway flocking mechanism
  - project under constant development; with large community of users
  - no economy; no trust/reputation;

NimrodG

- Utilizes an economy-driven architecture for managing resources and scheduling jobs
  - uses existing services provided by the Grid middleware systems such as Globus, Legion, Condor; as well as GRACE trading mechanisms
  - basic scheduling algorithms:
    - Time Minimization, within time and budget constraints
    - Cost Minimization, within time and budget constraints
    - None Minimization, within time and budget constraints
  - latest version of this middleware (v3.0.1) was released in October, 2005; work from 2007 (at the WWW site)

ARMS (1)

- Uses agents for resource advertisement and discovery at the Grid global level
  - Agents are homogeneous and cooperating and are organized in a hierarchical structure
  - based on utilization of performance prediction capabilities provided by the Performance Analysis and Characterize Environment (PACE) toolkit
  - scheduling driven by QoS requirements of job requests (users have to specify an explicit job execution deadline)

ARMS (2)

- Agents act as representatives for local Grid resources
  - for each resource, PACE is used to create a hardware characterization template
  - hardware model and services information of each Grid resource are advertised across the agent hierarchy
    - information periodically updated using push and pull
  - information utilized to build knowledge-base named the ACT (Agent Capability Table)
  - performance model from the ACT and user requirements used to schedule job execution
- Project is dead since approximately 2001

JADE extension by Poggi et.al. (1)

- Two approaches to Grid-Agents integration:
  - to extend the Grid middleware to support agent features
  - to extend agent-based middleware to support functionalities of the Grid
- JADE extensions
  - mechanisms for code distribution, reconfiguration, goal delegation, load balancing optimization and QoS definition
  - new types of agents to support
    - rule-based creation and composition of tasks
    - mobility of the code at the task level (i.e., JADE behaviors or simply rules are exchanged by agents)

JADE extension by Poggi et.al. (2)

  - Drools agent to receive and execute rules coming from other agents (all use Drools open-source rule engine)
  - BeanShell agent to receive and execute behaviors coming from other agents
    - integrates the BeanShell scripting engine into the systems → a wrapper agent
- Work originally reported in 2004; since then two papers produced; no Grid related add-on listed among JADE add-on software; project seems to be a zombie...

Bond

- Java-based middleware for network computing
  - developed to create an infrastructure for scheduling complex tasks and data annotation for data intensive applications
  - knowledge and workflow management based on distributed objects
    - resource information is stored in language-based distributed objects
  - KQML for information exchange
  - distributed awareness is used to learn about existence of other agents
- Publications appeared in the 1999-2003 time frame → project is dead

Agent-based Scheduling Framework ASF (1)

- Main idea → to reduce the responsibilities of a conventional metascheduler
  - workload of the metascheduler grows with the number of computing resources grows (degrading scalability and accuracy of scheduling)
  - accuracy of scheduling degrades in overloaded metascheduler (not all information will be processed in time to make accurate prediction)
  - framework needed for managing a large number of heterogeneous computers, in multiple administrative domains with various operation policies combined in a Grid as a VO

Agent-based Scheduling Framework ASF (2)

- Composed of a metascheduler and autonomous agents attached to each computing resource manager
- Agents autonomously find jobs; instead of being assigned a job by a higher-level metascheduler
  - idea dual to conventional metaschedulers
    - conventional metaschedulers push jobs to resources
    - in ASF agents discover jobsand receives them the metascheduler (a pull-based approach)
- ASF prototype implemented replaced CSF in Globus; in 3-node environment 11% improvement of elapsed job processing time

MAGDA (1)

- Mobile Agent based Grid Architecture (MAGDA) was designed to address problems:
  - lack of ability to migrate an application from one system to another
  - low level of abstraction of the heterogeneity of the environment
  - lack of mature fault tolerance features
  - existing frameworks do not scale to the Grid level, or are focused on specific aspects
  - lack of support of task migration, monitoring and execution (with adequate checkpointing)

MAGDA (2)

- MAGDA supports
  - resource discovery
  - performance monitoring and load balancing
  - task execution within the Grid
- Layered architecture following the Layered Grid Model
  - should be possible to integrate MAGDA components with other Layered Grid frameworks
- Service discovery performed using Web Services
- Application-level load balancing provided by static and mobile agents

Which model can work? (1)

- Three basic approaches to the problem of resource management/brokering:
- ASF approach – LMaster is passive, while Workers are actively pulling jobs to be executed
  - PROBLEM: how to deal with complex SLA's
    - assume that a high priority and high paying job is contracted; it should be started immediately (and on the right hardware-software combination
    - there seems no way to assure that this will happen – Worker agents are autonomous and they(!) are active

Which model can work? (2)

- Negotiations LMaster ↔ Workers
  - use the same model as for User ↔ LMaster negotiations
  - LMaster issues a CFP; Workers respond; best contract is selected
  - interesting as it moves adaptive economic model deeper into the system
  - more complex as it adds one more layer of negotiations → Workers have to keep and mine market data; negotiating instead of working...
  - PROBLEM: how to deal with complex SLA's

Which model can work? (3)

- LMaster as an omnipotent manager
  - has knowledge about state of the system
    - Workers can report their state / load in predefined intervals
  - knows hardware / software characteristics of all Workers
  - knows details of all contracts
- What about scalability?
  - Scheduler overload problems have to be addressed
  - Response → system is adaptive → each new LMaster should be better than the previous one → LMaster accepts only as many Workers as it can manage (or team will be killed by “trust-effect”)

Concluding remarks

- Seems that most “existing” agent-based approaches
  - do not exist anymore (e.g. ARMS, Poggi, Bond)
  - are currently focused on other aspects (e.g. MAGDA)
  - may have problems with complicated SLA's (e.g. ASF)
- Our answer – an ecosystem of agents each with its own function – task provider, scheduler/manager or worker
  - scheduler agents having full control over its Workers
  - size of the team adaptive to the capacity of the manager

Thank you!

- Questions?