Utilizing agent teams in grid resource management—preliminary considerations

Mateusz Dominiak
Department of Mathematics and Computer Science, Warsaw University of Technology

Wojciech Kuranowski
Wirtualna Polska, Software Development Department

Maciej Gawinecki
Systems Research Institute, Polish Academy of Science

Maria Ganzha
Department of Administration, Elbląg University of Humanities and Economy
ganzha@euh-e.edu.pl

Marcin Paprzycki
Computer Science Institute, SWPS

Abstract

Recently it was suggested that (mobile) software agents can provide an infrastructure for resource management in grids. In this note we introduce an approach based on agent teams, and discuss how it can be used in grid resource management. Details of initial implementation of one of its functionalities are discussed.

1. Introduction

Grid computing has emerged as a promising approach to utilizing heterogeneous, geographically distributed, multi-domain computer resources. Virtualization of computing resources by grid computing is expected to provide its users with highly available and adaptable computing utilities. It is also expected to have a broad impact in science, businesses and industries. Unfortunately, the uptake of the grid, while speeding-up recently, is still unsatisfactory. One possible reason for this situation is an overly complicated support for resource management provided by current grid software infrastructure.

At the same time it has been suggested that software agents combined with ontologies may provide the necessary infrastructure, by infusing grid with intelligence [5, 12]. Accepting arguments presented there, we have searched for the existing solutions that match this vision. While the results of our search are summarized in the next section, we can say that in our view all found solutions were somewhat limited in scope and not robust enough. Therefore we propose a different approach that is based on agent teams that collaborate to fulfill user requirements. Due to the lack of space we are only able to introduce the general framework of our proposed solution. In the next section we briefly summarize the state of the art in agent-based grid resource management. We follow with the description of our system. We complete this note presenting how we implemented a resource discovery service.

2. Agents in grids today

The initial work on agents in grids can be traced at least to [2], where J. Cao and colleagues adress the question of resource discovery in grids. They propose a hierarchical agent-based structure and experimentally evaluate various optimizing strategies for information distribution. Obviously, while very interesting, this work addresses only a small sub-area of usage of agents

in grids. Furthermore, the proposed framework was to be anchored in the PACE infrastructure, which by now seems to be extinct.

More recently B. Di Martino and O. Rana have proposed MAGDA (Mobile AGent Distributed Application), a mobile agent toolkit designed to support (1) resource discovery, (2) performance monitoring and load balancing, and (3) task execution within the grid [4]. Here, a dedicated mobile agent visits servers in the grid and collects local system information that is used to optimize distribution of application workload among agents or to move an agent from a heavier loaded node to a less loaded one (computational tasks are carried by mobile agents to nodes where they will be executed). However, the proposed system does not have an economic model associated with it. Furthermore, it was implemented in Aglets which, though recently becoming an open source product, seem to be slowly turning into a historical reference.

Last year S. S. Manvi and colleagues proposed somewhat different approach to agents in grids [9]. They started from an economic model and utilized mobile agents which traverse the network to complete a user defined task. At each visited node agents finds out what are local conditions for job execution and if acceptable, execute their job there (if they are not, they move on). In their work, among others, authors consider a number of pathway selection scenarios.

Also last year, Ouelhadj and colleagues considered negotiation (and re-negotiation) of a Service Level Agreement between agents representing resources and resource users [10]. Negotiations were to be based on Contract Net Protocol, however their paper was focused on higher level functionalities of the system. Again, this work considers only a specific sub-area of utilization of agents in grids.

While interesting, we can see some problems with the proposed approaches. (1) Most of them are limited in scope and functionality and do not involve economical foundations. (2) Some of them rely on agent mobility, while not considering its cost—since agents carry tasks, their size depends on the size of transported code and data and thus agent mobility should be used very judiciously. (3) Proposed infrastructures do not take into account full effect of grids highly dynamic nature and use single service providers—this leaves users vulnerable to (sometimes) rapid fluctuations of workload of individual nodes, as well as nodes disappearing and reappearing practically without warning. (4) Finally, reliance on "barely known" service providers should involve trust (reputation) management. To address these issues we propose solution that is based on utilization of agent teams.

3. Proposed approach

Let us start from very basic considerations. Computational grid can be viewed as an environment in which workers (in our case *agent workers*) that want to contribute their resources, and be remunerated for their usage, meet and interact with users (in our case *agent users*) that want to utilize offered services to complete their tasks. Obviously, we assume that at any moment worker may turn into a user and vice-versa. To be able to successfully facilitate needs of workers and users we propose an agent-centered infrastructure that is based on following general assumptions:

- agents work in teams (groups of agents)
- each teams has a single leader—*LMaster agent*
- each *LMaster* has a mirror *LMirror agent* that can take over its job in case when it "goes down"
- incoming workers (*worker agents*) join teams based on individual set of criteria
- teams (represented by their *LMaster*s) accept workers based on individual set of criteria
- decisions about joining and accepting involves multicriterial analysis (performed by so-called *MCDM modules*)
- each *worker agent* can (if needed) play role of an *LMaster*
- matchmaking is provided through yellow pages [13] and facilitated by the *CIC agent* [1]

Combining these assumptions we can develop system represented in Figure 1 as a Use Case diagram.

Let us now describe briefly dynamic processes that are depicted in their static form in Figure 1. To do this let us assume that the system is already running for some time, so that there already exist agent teams and their "advertisements" (both describing what resources they offer and/or what jobs they would like to execute; as well as what "types" of agents they would like to join their team) are posted with the *CIC* (while currently we use a single *CIC* in the future we may utilize an approach similar to that reported in [2]). First, observe that the *User*, represented in Figure 1, can either be someone who tries to contribute services to the grid, or someone who would like to utilize services available there. Interestingly, the Use Case diagram shows that both situations can be modeled in a "symmetric" way. Let us start from the case of *"User-contributor"* (processes that take place here are very similar to these described in [1] that provides further details).

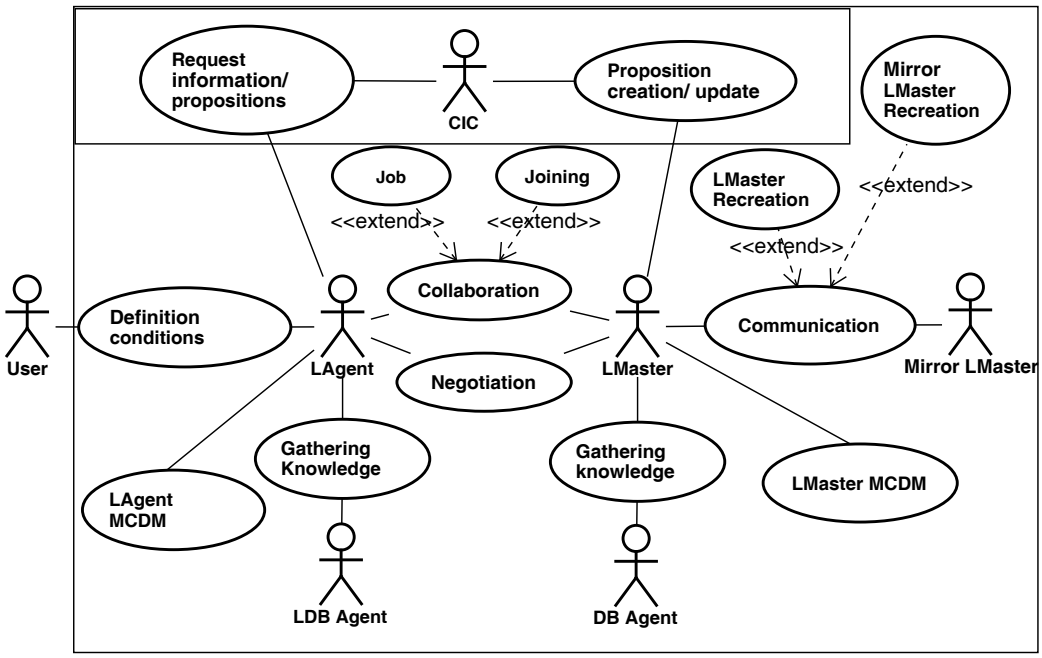

Figure 1. Use Case diagram of the proposed system

User who wants to contribute resources to the grid communicates with its agent (the local agent *LAgent* which, starts playing role of a worker agent) and formulates conditions for joining an agent team. Note that she may also request that a new team is to be created and her *LAgent* is to become its *LMaster*. The *LAgent* requests from the *CIC* list of agent teams that satisfy its predefined criteria. Upon receiving such a list, due to trust considerations (see [6] for more details) it may remove certain teams from the list. For instance, if it worked with a given team in the past and was "unhappy" with "rewards," it may not want to work with it again. For all the teams remaining on the list, the *LAgent* communicates with their *LMasters* utilizing Contract Net Protocol based negotiations and Saaty-type multicriterial analysis to evaluate obtained proposals. The result of interactions between the *LAgent* and *LMaster*s may be twofold: (1) it finds a team that it would like to work with and joins it, (2) no such team is found (either it is not interested in any offer from *LMasters* or no *LMaster* sent it an offer). In this situation the *LAgent* may decided to abandon the task and inform about it its *User*. It is also possible that the *LAgent* decides to become the *LMaster* of a new team itself. In this case, it prepares an offer describing (1) who it would like to invite to join its team, and (2) what resources it can contribute (what kind of jobs it is willing to work on); and send these two "advertisements" to the *CIC* to be "posted."

Let us now consider what happens when the *"User"* requests that its *LAgent* arranges execution of a task and specifies conditions of that operation (e.g. maximal price or maximum time length during which it has to be completed). As mentioned above, in this case the scenario is very much the same. The *LAgent* queries the *CIC* to find out which teams can execute its task. Upon receiving a list of teams that match the query, the *LAgent* may adjust it by removing from it these teams that cannot be trusted. It then communicates with the remaining teams (their *LMaster*s) and negotiates the best place to execute its job. It is assumed that Contract Net Protocol based negotiations (similar to these described in [10]) and multicriterial analysis of available offers will be utilized. Specifically, a well-known Saaty-approach will be utilized here to compare offer details (e.g. price, time, resources, warranty etc.), as it is one of the best methods to deal with quantifiable multicriterial decisions. Note that if no team will satisfy conditions imposed by the *User* then no deal will be reached. In this case the *LAgent* will report this situation to its *User* and await further instructions.

Let us now describe the relationships between the *LMaster* and the *LMirror*. When a new team is created, then the "founding agent" becomes its *LMaster*. The first agent that joins that team becomes the *LMirror*

(agent that can take over the team-lead in case when anything happens to the *LMaster*). Subsequent agents joining the team will become *worker agents*. We have not decided yet if the *LMirror* should be also working as a *worker agent* or if its role should be limited to mirroring the *LMaster*; this decision will based on experimental analysis of *LMirror*s workload and will be performed when the initial feature-complete version of the system will be implemented. The *LMaster* and the *LMirror* share all information that is pertinent to the existence of the team; e.g. list of workers and their characteristics, list of tasks that have been contracted and have to be executed, knowledge base that stores information about all past interactions with incoming users etc. It is assumed that the *LMaster* and the *LMirror* check each-others existence regularly in short time intervals. In the case when the *LMaster* does not respond to a ping-type ACL message the *LMirror* contacts the agent environment infrastructure (*Directory Facilitator* service) to check the status of the *LMaster*. If the *LMaster* is "gone" it takes over its role. Its first action is to promote one of worker agents to become its *LMirror* and pass to it all necessary information. Then it informs all necessary agents about the change (the fact that it is now the *LMaster* of the team). Similarly, the *LMaster* upon finding that the *LMirror* agent is "gone" immediately promotes one of worker agents to its role and passes to it all necessary information. In both cases, promotion of a worker to a role of an *LMaster* or an *LMirror* may require dealing with the task that the selected worker was executing at the time of its promotion. Let us note that the proposed solution is not bullet-proof. It is conceivable that both the *LMaster* and the *LMirror* will go down "almost simultaneously" (e.g. the *LMaster* realizes that the *LMirror* is gone, but before it promotes one of its workers to become its new *LMirror* it will go down itself) and thus the team will be "destroyed." However, such a situation should be relatively rare and our goal is not to create a completely bullet-proof infrastructure. Rather, our aim is to provide the proposed infrastructure with a reasonable level of resilience against common failures. Obviously, in a production environment further levels of defense against team destruction would have been developed.

Finally, let us briefly mention a few additional objects that appear in Figure 1. The *Gathering knowledge* functions denote collection of information about processes happening in the system. The *LMaster* collects information about all interactions with incoming task-carrying agents as well as about members of its team. In this way it may later decide to not to interact with certain clients or remove certain workers from its team. Similarly, the *LAgent* collects knowledge about what happened when it utilized services of various teams, as well as when it was a worker for various teams. Interestingly, since *Lagent* can play any role in the system, it is quite possible that an *LMaster* will turn into an *LAgent* who represents its *User* trying to find location to execute its task. Will it turn to its own former team to do it? Questions like this are going to be answered within the *LAgent MCDM* module and the *LMaster MCDM* module.

4. *LAgent — CIC* interactions

For the remaining part of this note let us focus our attention on how the interactions between the *LAgent* and the *CIC* have been implemented. In particular, we will discuss the interactions that take place when the *LAgent* is querying the *CIC* where to execute its task.

We have assumed that data in our system is to be stored in semantically demarcated form. In this context, an ideal situation would be if there existed an all-agreed "ontology of the grid." Unfortunately, while there exists a number of (separate and incompatible) attempts at designing such an ontology, at this stage they can be treated only as a "work in progress." Therefore we focus our work on designing and implementing agent system skeleton, while using a simplistic ontology. Obviously, when the grid ontology will be agreed on, our system *will be ready* for it. Currently, our ontology of grid resources is focused on their "computational" aspects, e.g. processor, memory and available disk space. What follows is a snippet of our OWL Lite based ontology:

```
:Computer
        :a owl:Class.

:hasCPU
        :a owl:ObjectProperty;
        rdfs:range :CPU;
        rdfs:domain :Computer.

:CPU
        :a owl:Class.

:hasCPUFrequency
        :a owl:DataProperty;
        rdfs:comment "in GHz";
        rdfs:range xsd:float;
        rdfs:domain :CPU.

:hasCPUType
        :a owl:ObjectProperty;
        rdfs:range :CPUType;
        rdfs:domain :CPU.

:CPUType
        :a owl:Class.
```

```
Intel :a :CPUType.
AMDAthlon :a :CPUType.

:hasMemory
        :a owl:DatatypeProperty;
        rdfs:comment "in MB";
        rdfs:range xsd:float;
        rdfs:domain :Computer.


:hasUserDiskQuota
        :a owl:DatatypeProperty;
        rdfs:comment "in MB";
        rdfs:range xsd:float;
        rdfs:domain :Computer.

:LMaster
  :a owl:Class;

:hasContactAID
        :a owl:ObjectProperty;
        rdfs:range xsd:string;
        rdfs:domain :LMaster.

:hasUserDiskQuota
        :a owl:DatatypeProperty;
        rdfs:comment "in MB";
        rdfs:range xsd:float;
        rdfs:domain :Computer.
```

Let us now assume that the *LMaster3* agent has in its team worker *PC2929* which has a 3.7 GHz Intel processor, 512 Mbytes of memory and 400 Mbytes of disk space available as a "grid service." In our ontology it would be represented as:

```
:LMaster3
  :hasContactAID
        "monster@e-plant:1099/JADE";
  :hasWorker :PC2929.

:PC2929
        :a :Computer;
        :hasCPU
        [
                a :CPU;
                :hasCPUType :Intel;
                :hasCPUFrequency "3.7";
        ] ;
        :hasUserDiskQuota "400";
        :hasMemory "512".
```

Ontologically demarcated data will be stored (by the *CIC*) in a Jena repository [8]. To query Jena persisted data we have decided to use the SPARQL language [11]. Let us now assume that the *LAgent* is looking for a machine with an Intel processor of at least 3.2 GHz, at least 256 MB of RAM, and at least 350 MB of disk space. Then the SPARQL query will have the form:

```
PREFIX : <http://www.ibspan.waw.pl/mgrid#>

SELECT ?contact
WHERE
{
  ?lmaster
    :hasContactAID ?contact;
    :a :LMaster;
    :hasWorker
      [
        :a :Computer;
        :hasCPU
          [ a :CPU;
            :hasCPUType :Intel;
            :hasCPUFrequency ?freq;
          ];
        :hasUserDiskQuota ?quota;
        :hasMemory ?mem;
      ].
  FILTER (?freq >= 3.2)
  FILTER (?quota >= 350)
  FILTER (?mem >= 256)
}
```

and the response that points to the above described machine would look as follows: `monster@e-plant:1099/JADE`. Specifically, it points to the *LMaster* that has that machine (worker) in its team. Obviously, a complete response would consist of a list of all teams that have among them at least one machine that satisfies the above described criteria.

When implementing the *CIC* we have decided to use multiple *database accessing agents* (*CICDB*). In this way we follow the results of our experiments reported in [3], where we have shown that using multiple DB agents can improve throughput of database access. Our system is being implemented in JADE [7] and in our current implementation we are using up to 5 *CICDB* agents that are utilized using a "push" method—the *CIC* knows which *CICDB* is free and pushes to it the next query to be handled. In Figure 2 we present results collected by the JADE *Sniffer agent* that depict interactions between *LAgent*s and the *CIC*. There we can see 5 different *LAgent*s (named user–agent(1–5)) and the *CIC* that utilizes its *CICDB* agents (named *ICDB0*, *CICDB1* and *CICDB2*) to respond to their queries.

5. Concluding Remarks

In this note we have introduced our agent-team-based approach to grid resource management. We have sketched the overall system design and presented in some detail the way that a resource query has been implemented. Currently we proceed with implementation of the complete agent-based skeleton of the system,

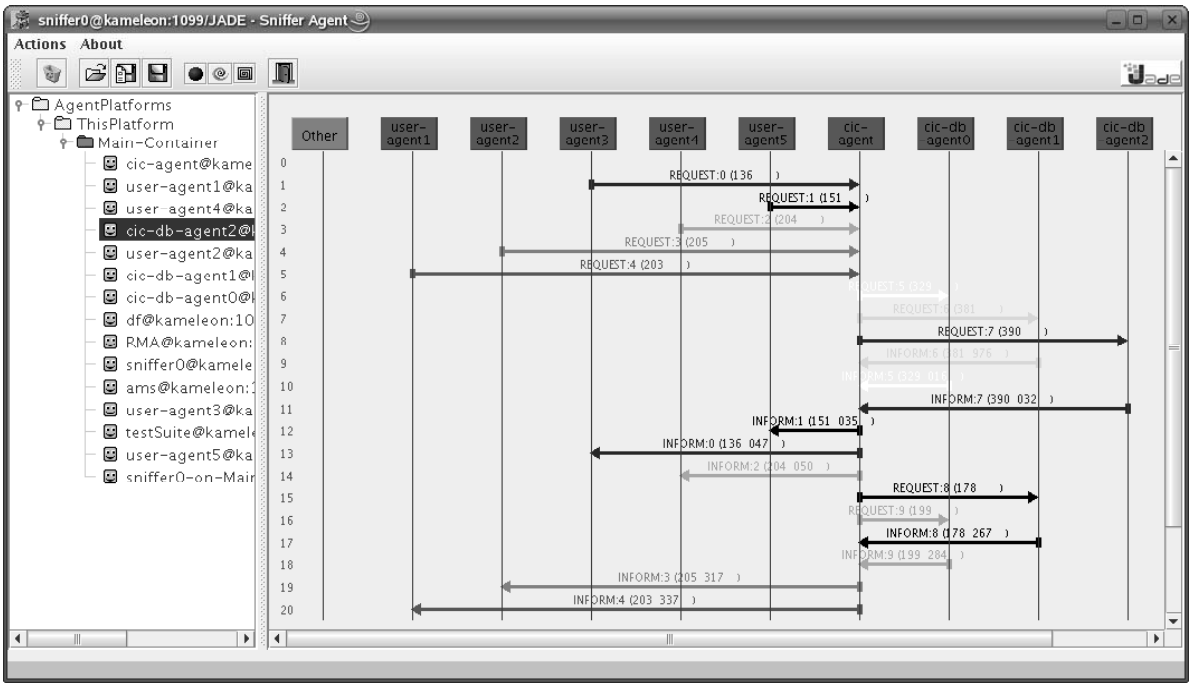

Figure 2. CIC and User interaction

while simplifying (though keeping as realistic as possible) its non-agent parts. An example of this approach is our utilization of an extremely limited ontology of grid resources, while utilizing robust tools like SPARQL and Jena to operate on it. Furthermore, while recognizing the importance of security, since this is a large research issue in its own right, we omit it in this stage of system development. We are implementing the system and will report on our progress in subsequent papers.

References

[1] C. Bádicá, A. Báditá, M. Ganzha, M. Paprzycki, Developing a Model Agent-based E-commerce System. In: Jie Lu et. al. (eds.) E-Service Intelligence - Methodologies, Technologies and Applications, Springer, in press

[2] J. Cao, D. J. Kerbyson, G. R. Nudd, Performance evaluation of an agent-based resource management infrastructure for grid computing, in: Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001, 311-318

[3] K. Chmiel, D. Tomiak, M. Gawinecki, P. Kaczmarek, M. Szymczak, M. Paprzycki, Testing the Efficiency of JADE Agent Platform. In: Proceedings of the ISPDC 2004 Conference, IEEE Computer Society Press, Los Alamitos, CA, 2004, 49-57

[4] O. F. Rana, B. Di Martino, Grid performance and resource management using mobile agents, in: Performance analysis and grid computing, 2004, 251-263

[5] I. Foster, N. R. Jennings, C. Kesselman, Brain Meets Brawn: Why Grid and Agents Need Each Other, AAMAS'04, July, 2004, ACM Press, 2004, `http://www.semanticgrid.org/documents/003-foster_i_grid.pdf`.

[6] M. Ganzha, M. Gawinecki, P. Kobzdej, M. Paprzycki, C. Bádicá, Towards trust management in an agent-based e-commerce system – initial considerations, in press

[7] JADE: Java Agent Development Framework. See `http://jade.cselt.it`.

[8] Jena—A Semantic Web Framework for Java. See `http://jena.sourceforge.net/`

[9] S.S. Manvi, M.N. Birje, Bhanu Prasad, An Agent-based Resource Allocation Model for computational grids, Multiagent and Grid Systems, 1(1), 2005, 17-27

[10] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, K. Krishnakumar, A. and Meisels, A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in Grid Computing. In: Peter M. A. Sloot et. al. (eds.), Advances in Grid Computing-EGC 2005, Springer-Verlag, 2005, 651-660

[11] SPARQL Query Language for RDF. See: `http://www.w3.org/TR/rdf-sparql-query`

[12] H. Tianfield, R. Unland, Towards self-organization in multi-agent systems and Grid computing, Multiagent and Grid Systems, 1(2), 2005, 89-95

[13] D. Trastour, C. Bartolini, C. Preist, Semantic Web Support for the Business-to-Business E-Commerce Lifecycle, Proceedings of the International World Wide Web Conference, ACM Press, New York, USA, 2002, 89-98