Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 337 – 344

ISSN 1896-7094

The Software Agents in a Database Grid

Marcin Gorawski[1], Sławomir Bańkowski[1], Michał Gorawski[1],

[1]Silesian University of Technology, Institute of Computer Science,
Akademicka 16, 44-100 Gliwice, Poland
{Marcin.Gorawski, Michal.Gorawski, Slawomir.Bankowski}@polsl.pl

Abstract. Software Agents methodology presented in the following paper is based on a grid structure. A single grid node can be adequately configured to access a remote folder or to remotely access other Agents. The tasks are distributed and executed parallel on many nodes. This causes a significant increase in the system performance. The build-in functions measureing a local node performance allows precise whole system efficiency monitoring. The system is adapted to a multi-user mode, and a good balancing mechanism provides equal use of all system nodes. It is essential for a full usage of a system capability in an unknown environment. The balancing objective is to omit slower nodes and to send more tasks to faster nodes

1 Introduction

The Software Agents (SA) realized in a grid technology are considered in [1, 2, 3, 4]. Despite rapid CPU evolution a wide range of their usage causes increase of demands for larger and faster computer systems. Answer to this problem are multiprocessor and multicore computers, however their price cannot compete with cheap single processor units. Efficient servers and units dedicated to specific solutions are very expensive and, on the other hand, software and data structure is not always compatible with parallel processing. The main problem is work distribution and load balancing. Load balancing can be optimal in an environment witch constant number of nodes of known characteristics [5]. In the grid systems an unknown number of nodes is available. The nodes are randomly logging on and off to network, so it's extremely difficult to obtain uniform workload on all nodes meanwhile minimizing distribution losses.

2 Software Agent and Agents system

The Software Agent is an autonomous unit working in a certain environment, capable of communicating, environment observation, decision making and learning, usage of gathered knowledge, adapting to environment, and error tolerant. All modules are implemented using joint interfaces, what enables uniform tasks and data swapping. The Agent was implemented in Java 1.5.

The *Management* module is responsible for data distribution inside the Agent. The *Security* module checks, if incoming tasks come from a legible source. The *Clients* module enables access to catalogues and other services via network, and the *Configuration* module allows setting and storing data important to the Agent. *Tasks execution* module is crucial for the system efficiency. Tasks are added to the Agent space and then serviced by appropriate modules. The *Database access* module assures management and maintaining of connections to

various databases (JDBC and ODBC usage). The Agent system is based on a grid technology. The shared access to all resources assures high scalability.

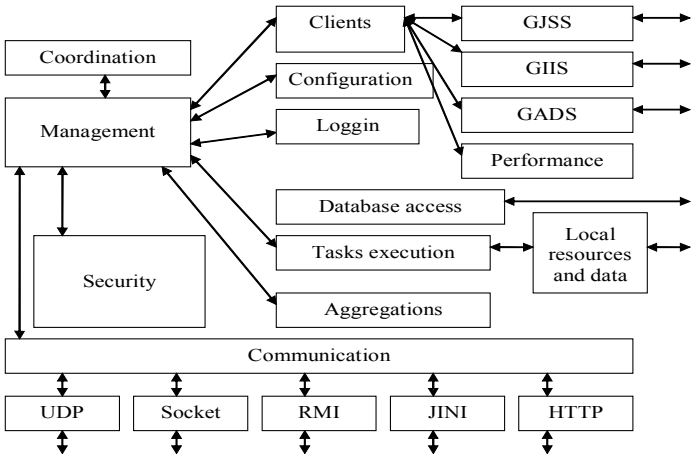

Fig. 1. Agent structure

3 Agent tasks

The task consists of a series of parameters with their values. Parameters names are optional. Task serialization stores a task in following manner

```
Parameter1=Value1;Parameter2=Value2;Parameter3=Value3;Parameter4=Value4;Parameter5=Value5;Parameter6=Value6;Parameter7=Value7
```

The task example:

```
workerGroup=math;workerFun=do;time=0;groupId=319225031;e0=5000000.5;version=3.2.0.0;endMode=r(p1);p2=1.8;p1=1.4;giis=false;num=0;gjss=true;workerMode=r(p1);b0=1.0;timeout=1116262256207;count=1;timeCreate=18:50:46;valueSum=3498326.3905740315;endGroup=math;group=workerQueue;id=757738465;endFun=add;fun=set;dimensions=1;splitParts=2;agentName=Agent11;
```

The Agent receives tasks from a catalogue and sends its tasks or other task's results. To control task execution three queues were created. Each task goes through all of them.

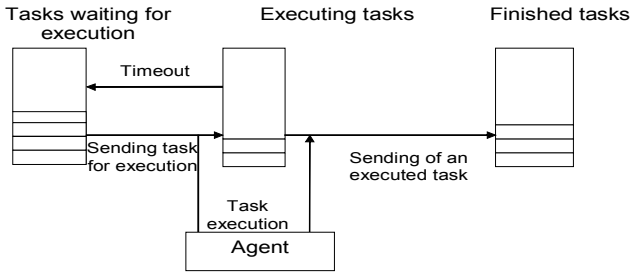

Fig. 2. Queues structure

Every task can be divided into some number of subtasks, however while the fragmentation is to small not all system nodes would be used. In the case of too large fragmentation significant efficiency decreasement will be observed. A task that is to be send is divided into subtasks and loaded into the first queue (subtasks waiting to be executed). From this queue subtasks are send to agents and simultaneously its copy is loaded into the second queue (subtasks currently processed). Subtask includes a timestamp determining when it was loaded into queue, to ensure execution time control. The execution time is checked periodically and if it is too long, a timeout is generated and subtask is loaded back to the first queue (subtasks waiting to be executed). After a successful

execution (execution report or result is obtained) subtask is loaded into the third queue (finished tasks). If both, the first and the second queues are empty, that means that the whole task was completed successfully and subtasks results can be combined.

4 Database Grid

The grid is a form of a distributed computation, that coordinates and shares: calculations, applications, data, memory and network resources. The above is obtained through dynamic changing and geographically distributed organization. The grid technology could be the key to solve highly complex computation problems. Computational grids enable wide range of sharing and coordinating of geographically scattered network resources. The sharing can be long or short term and can concern heterogenic resources. This causes creation of dynamic virtual structures. As a result, users will not have knowledge of resources that are in the virtual structure. Computational grid defines scalability of a large number of network resources. Important question is resources competition, what determines grid efficiency. It is important to effectively deal with this problem, for example through monitoring and resources scheduling. Agents become part of a larger system through the grid technology. The communication and tasks execution is transparent to the user. Connections structure and resources are not important from the system user point of view

The network transfer speeds constantly increase, so it is more and more profitable to send tasks to less loaded nodes, execute them, and return the results. One of the main problems is when data set size is not adequate (considering task execution time), to be send to other node for execution, or even to be distributed over several nodes and executed. Another problem is a choice of an adequate workstation. The GJSS, GIIS and GADS are information catalogues. They use common dataspace available for every logged agent. Functionality is limited comes down to collecting and sharing information service which is clear for all agents. As an example every agent can load information about its efficiency, but also ask about most efficient agent in the same catalogue. GJSS, GIIS and GADS catalogues differs mainly in communication mechanisms (RMI, JavaSpaces, Socket).

5 Communication and distribution

A good communication is crucial for effective operation of distributed system. It must enable sending various information in an unknown Agent structure. Figure 3 shows various types of Agent connections. Grid structure stores information about agent resources, available indexes, efficiency and plug-ins. Agent can send its information or obtain information from other agents. These information is crucial for choosing an appropriate agent.

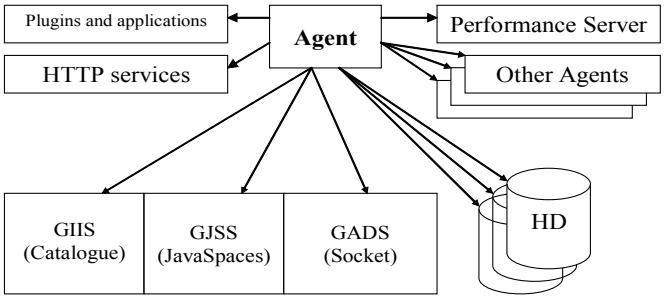

Fig. 3. Agent's communication

5.1 Service catalogues

5.1.1 GIIS

The GIIS is an universal catalogue storing information about Agents (Fig. 4). It also performs a role of a security system. The GIIS can store: information about reliable Agents, data about database connections, authorization sets and task designed for execution. When connecting to the GIIS, the Agent has to be sure, that this address is reliable. If the connection is correct, GIIS registration process is performed. This process involves: adding an Agent to a set, blocking Agent deletion and sending all required information (name, IP, diagnostic information).

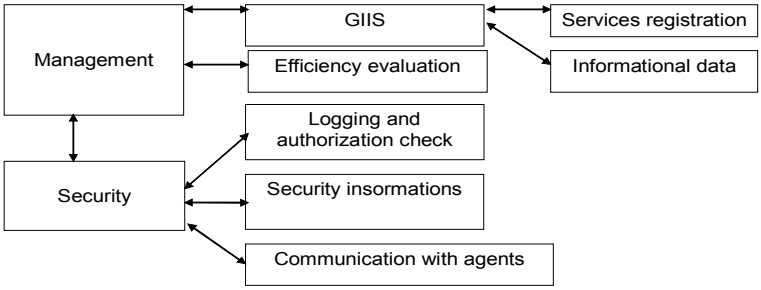

Fig. 4. GIIS schema

GIIS offers several functions necessary for data sharing and Agent efficiency evaluation. Registered Agent can upload list of all Agents, data of a chosen Agent or overall statistics

5.1.2 GJSS

The GJSS is a catalogue of information and Agents tasks and it is implemented using JavaSpaces technology (Fig. 5). Transactions mechanisms enable easy tasks management.

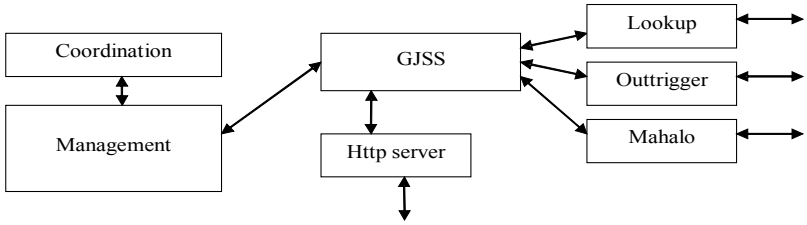

Fig. 5. GJSS schema

JavaSpaces creation consists of 4 stages:

- startup of the HTTP server, which provides access to jar files,
- startup of the Lookup service, which enables communication of grid clients and JavaSpace,
- startup of the Outtrigger service, which creates JavaSpaces spaces,
- startup of the Mahalo service, which supervises transactions.

5.1.3 GADS

The GADS is an Agent services catalogue (Fig.6). It is implemented using sockets. The Catalogue can have many clients, which registers during first connection. The AgentTask tasks are sent to this catalogue.

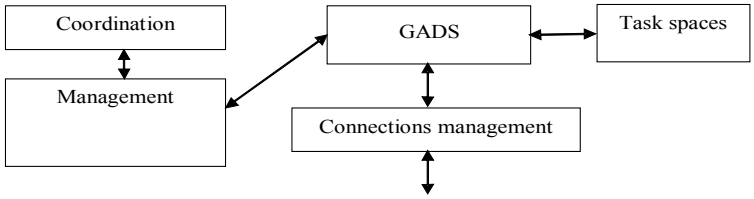

Fig. 6. GADS schema

The Catalogue has its own task spaces of a certain type. During registration an Agent creates its private Agent type space that can store Agent's tasks.

- agent – the Agent's private space, task added to the space is immediately send to a certain Agent,
- all – The added task will be immediately send to all Agents registered in the GADS,
- one – The added task will be immediately send to random Agent,
- private – The Agent's private space, where added tasks wait to be serviced,
- room – The tasks are stored in the FIFO order; if any Agent is free next task is send,
- status – Tasks are added to a hash table, where key is a gadsKey parameter, any Agent can collect all tasks without deleting them.

Tasks spaces in GADS catalogue enables automatic tasks send in many modes like all or address mode and also assures send to fastest agent, one or several agents or to first free agent.

5.2 Virtual Organization

Figure 7 shows Agents communication in a hierarchical structure. The VO is a virtual organization, where Agents swap tasks without accessing the catalogue. The VO is a set of resources creating a logical unit.

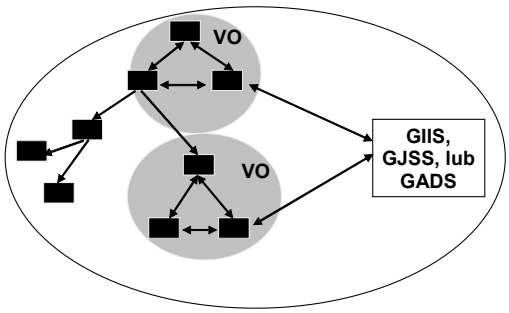

Fig. 7. Virtual organization

When a small number of workstations is accessible, one VO can be created. In such a case every Agent will have other Agents active connections. Communication is optimal, but balancing is complicated. When a new task is added, it is divided into several parts (usually five or six times more than the number of available Agents), and then it is distributed to all Agents. When the results from a certain Agent are sent, Agent receives a new task. In such a case, Agents' usage is full during the whole computation cycle, except for a short time period (20-80 miliseconds) when Agent waits for a new task. This time period is between tasks result sending and receiving a new task. During this time the agents cannot perform any actions, however the tests show that the time period is quite short and has no impact on efficiency. The problem can be easily solved by sending couple tasks to each Agent.

5.3 Direct and catalogue send

In case of the direct sending, the Agent uses sockets technology and swaps tasks directly. This is not suitable when the system consists of several nodes. The catalogue sending does not require additional time (serialization), moreover transactions are not serviced, and for one task there is only one sending (Fig.8).

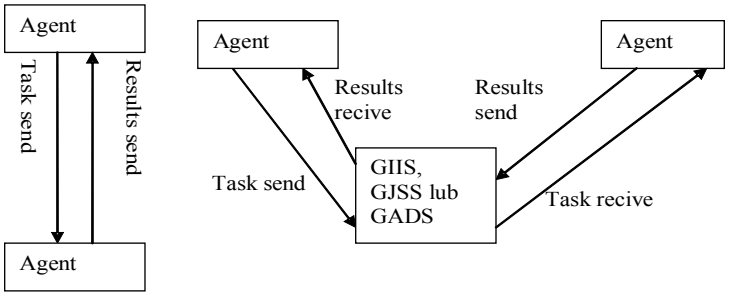

Fig. 8. Direct and catalogue send

6 External plug-ins

In a situation when an Agent operates on hundreds of computers, it is quite frequent that function required for certain computation is unavailable. The Java language assumes interoperativity which enables dynamic loading of classes. The dynamic loading allows implementation of a one class (according to interface) and sharing it with all Agents. In this manner a certain class is quickly distributed in the system and the task is ready to be processed in a parallel fashion.

When using dynamic class loading security problem emerges. The information is needed which Agents shares certain class and if this class can be safely used. So there has to be a reliable module that will accept data only from authenticated sources. The solution is a GIIS catalogue in which task of a new class is send to all Agents simultaneously, so the new class is availibe to every agent a few seconds.

Implementation of a new plug-in means implementation of a class according to the interface:

```
public interface AgentOuterPluginI {
  /** Plug-in start*/
  String start(String s);
  /** Plug-in stop*/
  String stop(String s);
  /** Task execution by plug-in, task: task.toString() */
  String doTask(String s);
  /** Task receive */
  String getTask(String s);
}
```

Class files are automatically loaded after the system startup, and can also be loaded during Agent operation. If a connection to a GIIS catalogue is active, automatic plug-ins actualization is performed. Such solution enables fast distribution of new capabilities among all Agents.

Agents are connected to catalogues which makes the communication transparent. A task is sent without any knowledge where its designated agent is or how the task will get there. The main drawback of catalogues is their time overhead while sending the tasks. In order to reduce the negative consequences of this problem agents are connected directly, creating virtual organization (VO). In such VO tasks swapping is fast but it is limited only to the nearest neighbors.

7 Tests

Tests concerns communication, tasks execution and plugins swapping in Agents System.

The first test proves, that time overhead while communicating using catalogue is relatively short.
The second test checks efficiency of tasks distribution, and proves that managing time (task creation, task dividing, task sending, results receiving) is insignificant.

The third test concerns plug-ins distribution. It shows that the presented structure can be easily upgraded with new abilities. Time required for new plug-ins sending and installation is in range of several seconds.

The first test concerns communication efficiency. The task was fragmented and sent through VO, using GIIS, GJSS or GADS. When the number of fragments increased, management time increased proportionally. The task concerned calculating sum of a certain mathematical sequence (Fig. 9).

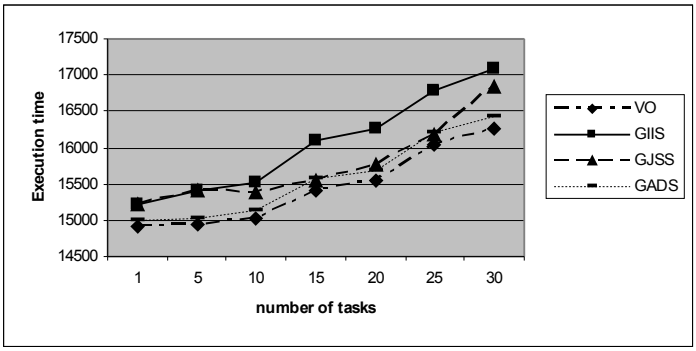

Fig. 9. Task sending

The speed-up of task execution equals the number of system nodes. When larger tasks are serviced results are considerably better (management time remains unchanged for a constant number of subtasks) than when executing the same task on a one station (Fig. 10).

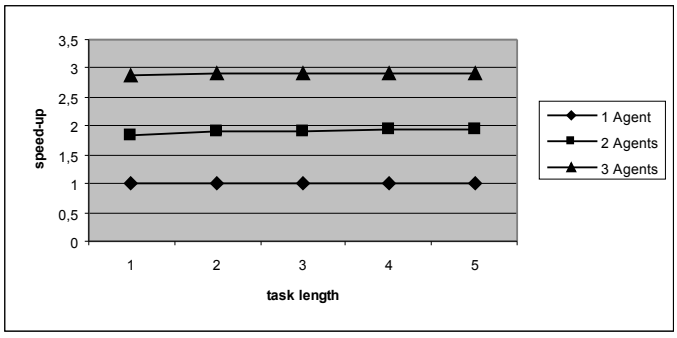

Fig. 10. Dependency of speed-up and task length

In a following test several classes distribution time was measured (Fig. 11). Plug-in is added to one Agent and time is measured between adding of a plug-in to one Agent and plug-in startup in all nodes.

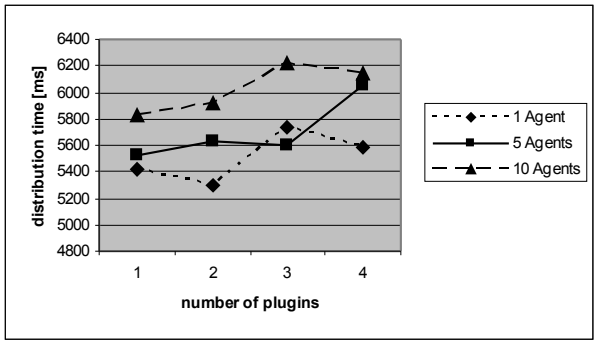

Fig. 11. Plug-ins distribution

8 Conclusions

The presented Agent methodology is based on the grid technology. Single node has the ability to configure, connect to a remote catalogue and directly to other Agents. Tasks are divided and executed parallel on many workstations. This results in a considerable speed-up. Built-in functions, measuring efficiency of a local node allows the whole system performance monitoring. The system is adapted to a multi-user mode, and a good balancing mechanism provides equal use of all system nodes. It is essential for a full usage of the system resources when working in an unknown environment. The goal of balancing is to omit slower nodes and to send more tasks to the faster nodes

In the case of clustered systems VO usage is recommended. The VO solves the Agent access problem. The usage of plug-ins increases system flexibility. As presented in the Software Agents system tests, tasks creation and parsing is quite fast, and a simple serialization allows data sending using JavaSpaces, RMI, Sockets or datagram.

References

1. Faruk Polat, Reda Alhajj: *A multi- agent tuple-space based problem solving framework,* Middle East Technical University, 06531 Ankara, Turkey 1998.
2. H. N. Lim Choi Keung, J. Cao, D. P. Spooner, S. A. Jarvis, G. R. Nudd, *Grid Information Services using Software Agents,* 2003.
3. Gorawski M., Bańkowski S.: *Software Agents in Grid Environment,* I National Scientific Conf. -Technology of the Data Processing, Poznań, pp. 118-129, 2005.
4. Jorge R. Bernardino, Pedro S. Furtado, Henrique C. Madera, *Approximate Query Answering Using Data Warehouse Stripping,* 2003.
5. Junwei Cao, Daniel P. Spooner, Stephen A. Jarvis, Subhash Saini and Graham R. Nudd, *Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling,* 2003.