In SIMULATION: Transactions of The Society for Modeling and Simulation International, Special Issue on Modeling and Simulation Applications in Cluster and Grid Computing, SAGE Publications, Vol. 80, No. 4-5, pp. 221-229, 2004.

# ARMSim: a Modeling and Simulation Environment for Agent-based Grid Computing

JUNWEI CAO
*C&C Research Laboratories, NEC Europe Ltd., Germany*
*cao@ccrl-nece.de*

**Abstract.** ARMS is an agent-based resource management system for grid computing, where agents are organized into a hierarchy and cooperate with each other to discover available grid resources using a technique of decentralized resource advertisement and discovery. Since a large-scale application of ARMS is not available, the most straightforward way to investigate the ARMS performance is through a modeling and simulation approach. In this work, an ARMS performance modeling and simulation environment (ARMSim) is presented. The ARMSim kernel is composed of a model composer and a simulation engine, while users can input related information and get simulation outputs from corresponding GUIs. A case study is included using an example model with over 1000 agents and several experiments are carried out each involving nearly 100000 requests. Simulation results are also illustrated and show the impact of the choice of different agent configurations on the overall system performance. ARMSim enables the ARMS agent performance to be investigated quantitatively and simulation results are potential to be utilized at running time for online ARMS performance improvement (e.g. avoiding performance bottleneck and reducing network traffic).
**Keywords:** modeling and simulation, multi-agent systems, grid computing, ARMS, ARMSim

## 1. Introduction

The grid originated from a high performance computing power delivery system [14] and is now becoming a global infrastructure for large-scale resource sharing and system integration [3]. The grid infrastructure should support both scalability and adaptability in dynamic grid environments. While examples of grid middleware technologies include CORBA [16], JINI [2] and Web Services [12], a multi-agent approach is utilized in our work.

In my previous work an agent-based methodology [6] is developed for building large-scale distributed software systems with highly dynamic behaviors. This has been used in the implementation of an agent-based resource management system for grid computing [7, 8], ARMS, where agents are considered to be the main high level abstraction of grid resources. Each agent can be registered with multiple grid resources and agents are organized into a hierarchy. Agents can also cooperate with each other and resource information is advertised and discovered along the agent hierarchy using different configurations.

Two applications of ARMS can be found respectively in the work described in [9] and [10]. In the first application, the ARMS implementation is integrated with multiple local grid schedulers. Each scheduler is responsible for load balancing of a PC cluster using a performance prediction driven method and an iterative heuristic algorithm [18]. The ARMS agents perform advertisement and discovery across multiple clusters to achieve overall load balancing at the grid level. In the second implementation, ARMS performs resource discovery for a grid workflow management system according to user QoS requirements. In both cases, the ARMS implementation is limit with up to 30 agents involved, which is far from a grid size. The performance of ARMS agents themselves can not be investigated in details due to the absence of a large-scale implementation.

In this work, a modeling and simulation environment for the ARMS agents (ARMSim) is developed, which can be used to evaluate the overall ARMS agent performance in a quantitative way according to some pre-defined metrics. ARMSim has as input all of performance related information of the agent system, it composes them into a performance model, simulates the resource advertisement and discovery processes step by step, and finally outputs all of the statistical data. ARMSim supports multi-view and real-time display of simulation results. Simultaneous simulation of multiple models and comparison of results can also be performed in the ARMSim environment.

A case study is included in this work with about 1100 agents involved. 13 experiments are designed, each with a different agent configuration. During each simulation nearly 100000 requests are sent out and in some situations nearly 10000000 communications are involved for resource advertisement and discovery. Simulation results show that agent configurations can have very different impacts on system performance and the simulation approach is the most straightforward way to enable the system performance to be investigated quantitatively.

The rest of the paper is organized as follows: In Section 2 an overview introduction of the ARMS system is given. In Section 3, the ARMSim structure is described in detail. A case study with corresponding simulation results is illustrated in Section 4. Related work is summarized in Section 5 and the paper concludes in Section 6.

## 2. ARMS

Agents comprise the main components in the ARMS system; the agents are organized into a hierarchy and are designed to be homogenous. Each agent is viewed as a representative of multiple grid resources at a higher level of grid management. The information of each grid resource can be advertised within the agent hierarchy (in any direction) and agents can cooperate with each other to discover available grid resources [8].

Each agent utilizes Agent Capability Tables (ACTs) to record grid resource information. An agent can choose to maintain different ACTs corresponding to the different sources of resource information: T_ACT is used to record information of an agent's own registered grid resources; C_ACT to record resource information cached during discovery processes; L_ACT to record information received from lower agents in the hierarchy; and G_ACT to record information from the upper agent in the hierarchy.

There are basically two ways to maintain the contents of L_ACT and G_ACT in an agent: data-pull and data-push, each of which has two approaches: periodic and event-driven. An agent can ask other agents for their resource information either periodically or when a request arrives. An agent can also submit its resource information to other agents periodically or when the resource information is updated. The frequency of the periodical resource information advertisement is also configurable.

Another important process among agents is resource discovery which is also a cooperative activity. Within each agent, its own registered grid resources (recorded in the T_ACT) are evaluated first. If the requirement can be met locally, the discovery ends successfully. Otherwise resource information in C_ACT, L_ACT and G_ACT is evaluated in turn and the request dispatched to the agent, which is able to provide the best (or the first) requirement/resource match. If no resource can meet the requirement, the request is submitted to the upper agent. When the head of the hierarchy is reached and the available resource is still not found, the discovery terminates unsuccessfully.

The ARMS architecture and mechanisms described above allow possible system scalability. Most requests are processed in a local domain and need not to be submitted to a wider area. Both advertisement and discovery are processed between neighboring agents and the system has no central structure, which otherwise might act as a potential bottleneck. This is further investigated in a quantitative way using the ARMSim environment.

Four metrics are defined to evaluate the performance of agent behaviors: discovery speed ($v$), system efficiency ($e$), load balancing ($b$) and success rate ($f$) [7]. The average agent resource discovery speed ($v$) during a certain period is calculated via the total number of requests ($r$) divided by the total number of connections made for the discovery ($d$). The average efficiency of the system ($e$) is considered as the ratio of the total number of requests ($r$) during a certain period to the total number of connections made for both the discovery ($d$) and the advertisement ($a$). The workload of each agent ($w$) is described as the sum of the outgoing ($o$) and incoming ($i$) connection times and the mean square deviation of all agent workloads is used to describe the load balancing level of the system ($b$). The success rate ($f$) is the ratio of successful resource discovery ($r_f$) to the total number of requests ($r$) during a certain period.

These metrics may conflict at most of the time, that is not all metrics can be high at the same time. For example, a quick discovery speed does not mean high efficiency, as sometimes quick discovery may be achieved through the high workload encountered in resource advertisement and data maintenance, leading to low system efficiency. It is necessary to find the critical factors of a practical system, and then to use the different configurations to reach high performance. This can be carried out efficiently using a modeling and simulation approach.

## 3. ARMSim structure

Performance evaluation of resource discovery in a large-scale multi-agent system like ARMS is a difficult task, especially when thousands of agents and tens of thousands of requests and communications are involved. Different configurations of agent behaviors on resource advertisement and discovery can make the overall system behaviors very complex. In this section, the ARMSim modeling and simulation environment is introduced.

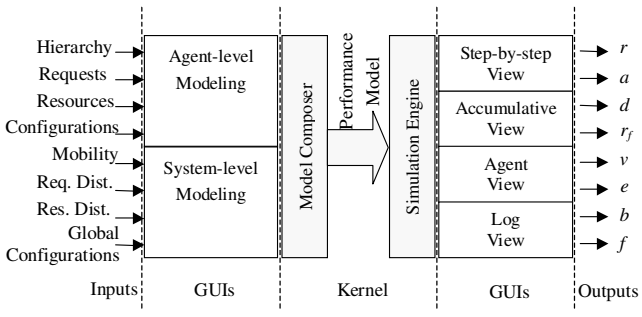

Figure 1. The ARMSim structure

The main ARMSim structure is illustrated in Figure 1, which includes a kernel and GUIs. The kernel part performs the modeling and simulation functions, while users can input related information and get simulation outputs from the GUIs.

### *3.1. Inputs/outputs*

There are four kinds of information that affect the system performance and must be input into the performance model. These include: the agent hierarchy, the resources, the requests, and the configurations for resource advertisement and discovery. ARMSim supports the modeling activity at both the agent level and the system level. The only components that exist in the model are agents, so agent-level modeling can be used to define all the model attributes for the simulation. However, system-level modeling is also necessary to input information on agent mobility, resource and request distribution, and so on. These will be discussed in detail below.

- *Agent hierarchy*. When a new agent is added into the model, its upper agent should be defined. The upper agent is also configured to add a new lower agent. The information is used to organize agents into a hierarchy in the system model. No cycles are permitted in the hierarchy, which may cause deadlock during the resource discovery process.
- *Requests*. Each agent is configured to send different requests periodically. A request item may include several parts of information: the required resource name, the relative required performance value, and the sending frequency (see examples in Table 3).
- *Resources*. Each agent is also configured to provide many grid resources, whose performance may vary over time. A resource item may include several parts of information: the resource name, the relative performance value, and the performance changing frequency (see examples in Table 2). The usage of these attributes will be introduced in the ARMSim kernel section below.
- *Configurations*. Different configurations are defined in each agent to control its behaviors on resource advertisement and discovery. These configurations have been introduced in Section 2 and examples can be found in Table 4.
- *Agent mobility*. The agent mobility can be defined at the system level only. An agent mobility item may include information on: the agent ID, the new agent ID after the movement, the upper agent ID of the new agent, and the step number when the movement will happen during the simulation.
- *Request distribution*. System-level request definitions can ease the modeling process. The same request item does not need to be defined in different agents one by one. ARMSim provides a convenient way to distribute a request definition to different agents once it is defined at the system level.
- *Resource distribution*. The same resource with the same attributes can also be provided by different agents. System-level resource definitions allow many agents to be configured with the same resource at one time.
- *Global configurations*. A system-level configuration definition can affect all of the agents in the model and ease the modeling process. Both global configurations and individual configurations can be defined in each agent. However, agent-level configuration definitions have a priority over the system-level ones.

The information above is input into the ARMSim. Examples of these information can be found in Section 4. The ARMSim outputs are all of the simulation results on four performance metrics. All of the details on resource advertisement and discovery are also recorded in a simulation log file for further reference. The use of input information to produce outputs during the modeling and simulation processes within the ARMSim kernel is introduced below.

### *3.2. ARMSim kernel*

The ARMSim kernel is composed of a model composer and a simulation engine. The kernel will perform the main modeling and simulation functions and transform the raw simulation data to statistical results to support the four performance metrics.

The model composer organizes the input information into a performance model before the simulation process begins. During this phase, the system-level information is transferred into an agent-level representation as much as possible. For example, system-level requests and resources will be used to configure a certain percentage of agents. The global configurations are used to define the configurations of each agent, except for agents that have already been defined with agent-level configurations. After these, a performance model is composed ready for evaluation. The information on agent movements can only be stored at the system level and will not be used to configure any agent in the system.

The simulation engine will start a simulation process once a performance model and a total number of simulation steps are defined. The whole process is illustrated in Figure 2, which is divided into seven phases, five of which are within the main simulation loop.

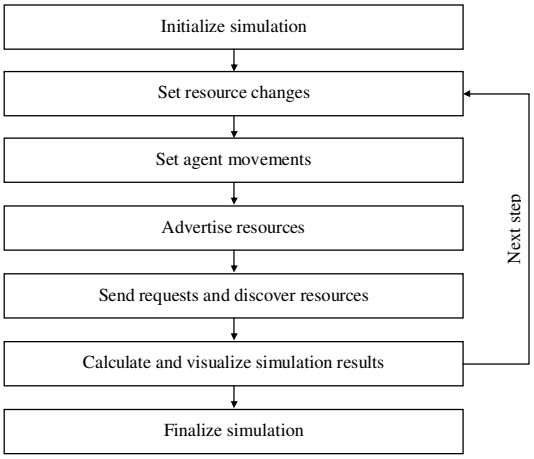

Figure 2. The ARMSim simulation process

- *Initialize simulation*. Once a simulation process is started, an environment will be setup for simulating resource advertisement and discovery. All of the GUIs for performance modeling are locked. The performance model cannot be modified during the simulation. The simulation results are also initialized for recording the outputs.
- *Set resource changes*. This is performed at the beginning of each simulation step. The performance of a grid resource may change at each step. There is also the frequency of change in performance of each grid resource. The performance of each resource may or may not be changed at each step according to this frequency.
- *Set agent movements*. Each agent mobility item contains a step number when a movement will happen during the simulation. An agent movement indicates not only the change of the agent hierarchy, but also the change of related resources. Additional resource advertisement occurs when an agent is moved, for example, old resource information is announced for deletion, and new information should be advertised along the new agent hierarchy.
- *Advertise resources*. Both event-driven and periodic resource advertisement are considered during this phase. Each agent acts on its ACTs according to its configurations. Each connection between agents for resource advertisement will be recorded in the simulation log file and will affect corresponding simulation results.
- *Send requests and discover resources*. A request is decided to be sent according to its frequency. Each agent that receives the request will look up its ACTs in turn according to its configuration for resource discovery. Every detail of a resource discovery process is recorded in the log file and related simulation results, such as agent connection times, are recorded.
- *Calculate and visualize simulation results*. At the end of each simulation step, the raw simulation data should be summarized, and corresponding statistical results on the performance metrics calculated. These results are shown on ARMSim GUIs dynamically to provide the user a view of what is going on during the simulation.
- *Finalize simulation*. After all simulation steps are completed ARMSim returns to the modeling mode. All the modeling GUIs are unlocked. The GUIs for visualizing the simulation results will not be refreshed until the next simulation begins, and can thus be used for further analysis.

ARMSim also supports the evaluation of multiple models simultaneously. The user can use different configurations in different models, simulate them, and compare the results.

### *3.3. User interfaces*

The ARMSim environment is implemented using Java. It provides graphical user interfaces for the modeling and simulation respectively.

The user can add, edit and delete agents from the model via the main GUI window (see an example in Figure 3). In the left column of the main window, all of the agents are listed. A brief description of the selected agent is also shown below the agent list. The text field above the agent list can be used to search an agent by its name. The model can also be saved and reloaded for reuse later.

Some other ARMSim GUIs are used to visualize simulation results to the user. Examples can be found in Figures 4 and 5. During each step in the simulation the results will be updated in each of the GUIs. The simulator can provide multiple views of the simulation data, which are all updated in real time. In the step-by-step view of the Figure 4, the simulation data, $r$, $a$, $d$, $r_f$, and the statistic data, $v$, $e$, $b$, $f$, in each step are shown. In the accumulative view shown in Figure 5, the statistical data on the accumulative steps are shown. In the agent view, the user can view the ACT contents of a selected agent. The log view shows the simulation log file, which records the details of all resource advertisement and discovery processes during simulation.

### *3.4. Main features*

ARMSim is developed to provide quantitative information of the performance of resource advertisement and discovery in the ARMS agent system. The main feature of the ARMSim environment can be summarized as follows:

- Support for all of the performance metrics and configurations described in the ARMS system;
- Support two levels of system modeling for easy and convenient performance modeling;
- Support modeling of agent mobility and simulation of additional resource advertisement processes;
- Support multi-view and real-time display of simulation results;
- Support simultaneous simulation of multiple models and comparison of results.

The use of the ARMSim environment for a performance study is introduced in the next section through a case study, and simulation results are included to show the impact of the choice of different agent configurations on the system resource discovery performance.

## 4. A case study

In this section, an example model is given and experiment results are included to show how to steer the ARMS performance optimization using the ARMSim environment.

### *4.1. Example model*

A screenshot of the example model built in the ARMSim environment is illustrated in Figure 3.

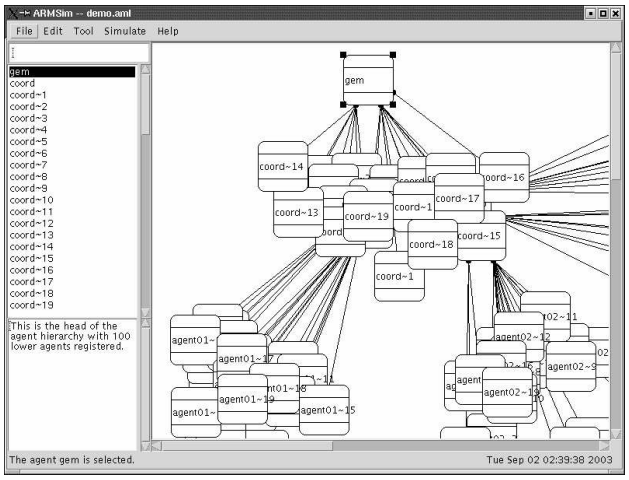


Figure 3. The ARMSim modeling

The attributes of the example model are shown in several tables. This is composed of about 1100 agents, each representing a one or more grid resources that may provide computing capabilities with different performances. These agents are organized in a hierarchy, which has three layers. The identity of the root agent is *gem*. There are 100 agents registered to *gem*, ten of which each also have 100 lower agents. The hierarchy is illustrated in Table1.

| Agents | Upper agent |
|---|---|
| *gem* | - |
| *coord~0.....coord~99* | *gem* |
| *agent01~0.....agent01~99* | *coord~5* |
| *agent02~0.....agent02~99* | *coord~15* |
| *agent03~0.....agent03~99* | *coord~25* |
| *agent04~0.....agent04~99* | *coord~35* |
| *agent05~0.....agent05~99* | *coord~45* |
| *agent06~0.....agent06~99* | *coord~55* |
| *agent07~0.....agent07~99* | *coord~65* |
| *agent08~0.....agent08~99* | *coord~75* |
| *agent09~0.....agent09~99* | *coord~85* |
| *agent10~0.....agent10~99* | *coord~95* |

Table 1. Example model: agent hierarchy

To simplify the modeling processes, we define the resources and requests in the agents at the system level, which is shown in Table 2 and 3 respectively. The name of the resources and requests are all HPC, but with different relative performance values. The frequency value of the resource, 5, for example, means the resource performance will change between 0 and the performance value once every 5 steps during the simulation. The frequency value of the request, 5, for example, means a request will be sent once every 5 steps during the simulation. The distribution value is used to define how many agents will be configured with the corresponding resource or request.

| Name | Relative performance | Frequency | Distribution (%) |
|---|---|---|---|
| HPC | 1000 | 5 | 20 |
| HPC | 800 | 8 | 30 |
| HPC | 600 | 10 | 40 |
| HPC | 400 | 15 | 50 |
| HPC | 200 | 20 | 60 |

Table 2. Example model: resources

| Name | Relative performance | Frequency | Distribution (%) |
|---|---|---|---|
| HPC | 100 | 5 | 80 |
| HPC | 200 | 8 | 70 |
| HPC | 300 | 10 | 60 |
| HPC | 400 | 15 | 50 |
| HPC | 500 | 20 | 40 |
| HPC | 650 | 30 | 30 |
| HPC | 800 | 40 | 20 |
| HPC | 900 | 50 | 10 |
| HPC | 1000 | 60 | 10 |

Table 3. Example model: requests

Finally, the model must define how each agent uses the ACTs to optimize the performance. In this case study six

experiments have been considered, each of which has the same configurations as described in Tables 1, 2 and 3, but has different configurations as described in Table 4.

| Configurations | Experiment number | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| Using T_ACT | X | X | X | X | X | X |
| Using C_ACT | | X | X | X | X | X |
| L_ACT: event-driven data-push | | | X | X | X | X |
| G_ACT: periodic data-pull* | | | | X | X | X |
| L_ACT: periodic data-pull* | | | | | X | X |
| G_ACT: event-driven data-push | | | | | | X |

*Here the frequency was once every 10 steps.

Table 4. Example model: configurations

To simplify the experiments, we only define the configurations at the system level, which means all of the agents in the model must use the same configurations. A mixture of configurations is possible but is not considered in these experiments. In the simulation results included in the section below, a comparison of the different configurations is given by considering their impact on the agent performance.

*4.2. Simulation results*

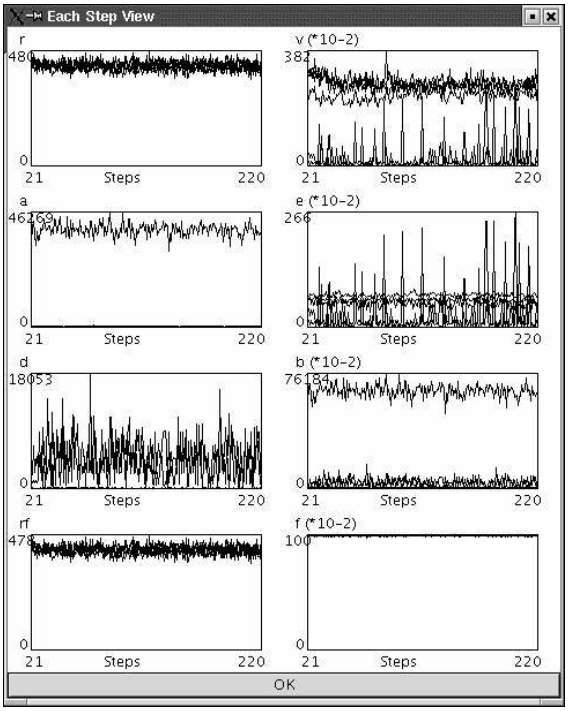


Figure 4. The ARMSim simulation: a step-by-step view

When the simulation begins, a thread is created to calculate the statistical data step by step. The phase for request sending and the resource discovery is the key part of the whole simulation process. The ARMSim environment can show the results in multiple views. The step-by-step and accumulative views are especially interesting in this case study, which is illustrated in Figures 4 and 5 respectively.

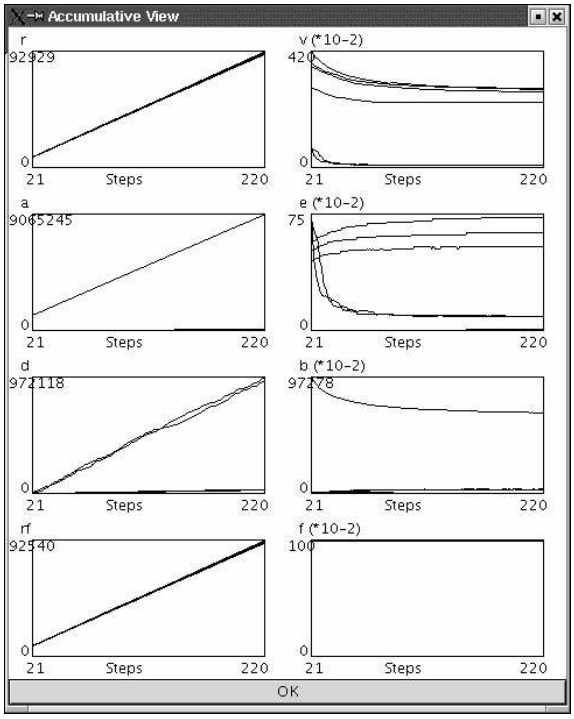


Figure 5. The ARMSim simulation: an accumulative view

As shown in Figure 5, six experiments are carried out simultaneously for 220 steps and results are displayed on all of performance metrics. Note that simulation processes in the first 20 steps are not stable and not included in Figures 4 and 5. We are especially interested in the balance of resource discovery speed and system efficiency in this case study. It is obvious that different configurations designed in these experiments lead to different impact on discovery speed and system efficiency. Detailed simulation data can be found in Table 5 and each of the six situations are also described below.

1. Only T_ACTs are used in each agent. Each time the request arrives, a lot of connections must be made and traversed in order to find the satisfied grid resource. In this situation, the discovery speed and system efficiency are both rather low.
2. The cache is used in each agent, which needs no extra data maintenance and improves the discovery speed and system efficiency a little. This is because the dynamics of the resources reduce the effects of the cached information and so becomes unreliable.

3. L_ACT is added in each agent. Each time the resource performance changes, the corresponding agent will advertise the change upward in the hierarchy. This adds additional data maintenance workload to the system, which decreases the discovery workload extremely. So the discovery speed and the system efficiency are all improved.
4. G_ACT is also added. Each agent will get global resource information from its upper agent once every 10 simulation steps, which will add additional data maintenance workload. From the simulation results, we can see this improves the discovery speed further. But the system efficiency decreases because of the additional data maintenance.
5. Another maintenance of the L_ACT is added. Each agent asks for resource information from its lower agents once every 10 steps. This improves the discovery speed a bit further and adds more data maintenance workload, which also decreases the system efficiency.
6. Another maintenance of the G_ACT is added. This improves the discovery speed only a little, but adds further data maintenance workload, which decreases the system efficiency extremely.

| No. | Performance metrics* | | | | |
|---|---|---|---|---|---|
| | *r* | *a* | *d* | *v=r/d* | *e=r/(a+d)* |
| 1 | 91848 | 0 | 972118 | 0.09 | 0.09 |
| 2 | 92326 | 0 | 849540 | 0.10 | 0.10 |
| 3 | 92206 | 89916 | 37583 | 2.45 | 0.72 |
| 4 | 92084 | 110648 | 34034 | 2.70 | 0.63 |
| 5 | 91264 | 138965 | 32415 | 2.81 | 0.53 |
| 6 | 92929 | 9065245 | 32837 | 2.83 | 0.01 |

*Note: All values are accumulative results after 220 steps.

Table 5. Simulation results I

The impact of the choice of the configurations on the discovery speed and the system efficiency is shown clearly in Figure 6.

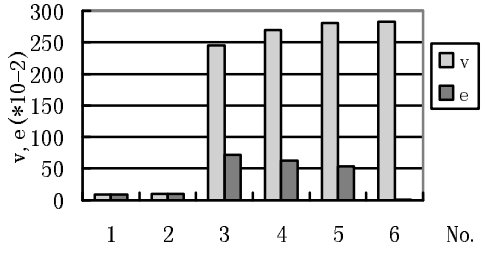


Figure 6. Choice of the configurations

It can be seen that both third and fourth experiments have a good balance between the discovery speed and the system efficiency for this example model. The fourth situation has a higher discovery speed in comparison to the third, with lower system efficiency. And the third situation has higher system efficiency with lower discovery speed.

The difference between configurations of the third and fourth experiments is that G_ACT periodic data pull is additionally configured in the fourth experiment, which was once 10 simulation steps. Changing the G_ACT data pull frequency will also change the performance of the model. Some further experiments are designed where the configurations that are used are all the same as described in the fourth experiment. The only difference is the G_ACTs in the agents are updated with different frequencies, which may lead to differences in the amount of system workload for resource advertisement and discovery. Detailed simulation results are given in Table 6.

| Freq. | Performance metrics* | | | | |
|---|---|---|---|---|---|
| | *r* | *a* | *d* | *v=r/d* | *e=r/(a+d)* |
| 1 | 91618 | 330256 | 32530 | 2.81 | 0.25 |
| 2 | 91537 | 210336 | 33346 | 2.74 | 0.37 |
| 5 | 91355 | 134910 | 33492 | 2.72 | 0.54 |
| 10 | 92084 | 110648 | 34034 | 2.70 | 0.63 |
| 20 | 90713 | 98893 | 33734 | 2.68 | 0.68 |
| 30 | 91641 | 95154 | 34935 | 2.62 | 0.70 |
| 80 | 92997 | 91803 | 35944 | 2.58 | 0.72 |
| 120 | 92540 | 88539 | 37016 | 2.50 | 0.73 |
| never | 92206 | 89916 | 37583 | 2.45 | 0.72 |

*Note: All values are accumulative results after 220 steps.

Table 6. Simulation results II

In Table 6, when the frequency value is once 10 simulation steps, the situation is the same as that in the fourth experiment. And when G_ACTs are never maintained, the situation is the same as that in the third experiment. The impact of the choice of the G_ACT periodic data pull frequency on the discovery speed and the system efficiency is shown clearly in Figure 7.

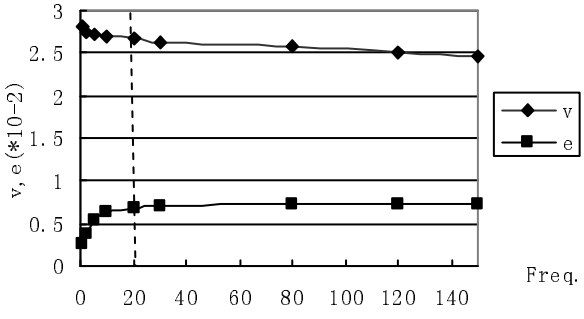


Figure 7. Choice of the G_ACT periodic data pull frequency

As shown in Figure 7, the best trade-off between discovery speed and system efficiency is once every 20

simulation steps in this example model. In summary, the example model should use all of the ACTs. L_ACT should be maintained by the event-driven data push configuration. The G_ACT should be maintained by the periodic data pull once every 20 simulation steps. In fact, the performance of the example model can be improved further using agent level modeling. Different agents can use a mixture of different strategies to achieve higher performance of the whole system. This is not discussed in detail here.

The techniques of modeling and simulation are useful especially for the current phase of research into grid computing. As mentioned, a practical grid environment does not yet exist. In fact, there is not even a grid testbed that can be used for research. Current ARMS application is far from a grid size, where the performance data cannot be produced for analysis. This makes a modeling and simulation environment very valuable for this kind of research. The ARMSim environment is such an attempt.

## 5. Related work

Modeling and simulation approaches have been widely used in grid computing research. Some existing tools are described below though motivations of these work are quite different from each other. Features that distinguish ARMSim from other work are also discussed in detail.

- *Bricks* [19]. The Bricks performance evaluation system allows analysis and comparison of various scheduling schemes in a grid computing environment. Bricks can simulate various grid behaviors, especially the behavior of networks and resource scheduling algorithms. Network modeling is not considered in current ARMSim implementation where reduction of number of agent connections is focused at the moment. ARMSim simulates the ARMS resource discovery process instead of resource allocation and scheduling.
- *GridSim* [4]. The toolkit supports modeling and simulation of heterogeneous grid resources, users and application models. It provides primitives for creation of application tasks, mapping of tasks to resources and their management. While ARMSim focuses on performance simulation of decentralized resource advertisement and discovery among ARMS agents, GridSim aims at design and evaluation of scheduling algorithms or policies of resource brokers. GridSim is based on an existing Java discrete event simulation infrastructure and the ARMSim simulation engine is an iterative sequential process relying on data statistical capabilities.
- *MicroGrid* [17]. The MicroGrid simulation tools enable Globus [13] applications to be run in arbitrary virtual grid resource environments. MicroGrid is actually an emulator meaning that actual application code is executed on the virtual grid. ARMSim takes a different approach and focuses on different issues. ARMSim characterizes ARMS agent behaviors using some statistical data like relative performance and frequency value and targets performance of the ARMS itself, while MicroGrid targets grid applications instead of performance of the Globus infrastructure.
- *Simgrid* [11]. Simgrid is a simulation toolkit for the study of scheduling algorithms for distributed applications. Simgrid targets scheduling algorithms for a single structured application. ARMS support resource advertisement and discovery for a multi-user system where all requests for computations are independent. Simgrid focuses more on application makespans as oppose to average overall performance.

Simulation approaches have also been used for performance studies of traditional high performance computing applications and systems for many years. Example simulation environments for parallel and distributed computing include POEM [1] and PACE [5, 15]. While ARMS focuses on grid level resource management, these work are not directly related to the ARMSim environment and thus not discussed here.

ARMS is an agent-based grid computing system with a generic hierarchical multi-agent model and a specific resource advertisement and discovery mechanism. The ARMSim environment is specially designed for the ARMS system. The benefit of an agent-based approach over other infrastructure techniques was discussed in [8]. The ARMSim environment is just initially implemented and future work is discussed below.

## 6. Conclusions and Future Work

This work addresses the problem of modeling and simulation of agent behaviors in the ARMS system. An initial implementation of the ARMSim environment is described in detail. A case study is included using an example model with over 1000 agents and 13 experiments are carried out each involving nearly 100000 requests. Simulation results show that agent configurations can have very different impacts on system performance and the simulation approach is the most straightforward way to enable the system performance to be investigated quantitatively in a large scale.

A major future work will be the refinement of input character models. Current ARMSim request and resource models are quite simple. Some parameters, e.g. performance and frequency, are modeled in an average way. In a real ARMS system, request distribution could be

very different and resource dynamics should also be characterized in a more refined way.

Since current ARMSim simulation can be processed quickly (in minutes given the example model described in Section 4), another future work would be the exploration of ways to integrate the ARMSim simulation engine with the ARMS system as an online performance advisor for ARMS. In this case, all input information should be monitored from the running ARMS agents and the ARMSim simulation results should be returned and used to advise ARMS agents with configuration suggestions that could lead to a higher overall system performance. For example, if agents are configured with more efficient resource advertisement and discovery, network traffic in the ARMS system can be significantly reduced. The ARMS agents can be also advised to move to a better location where more requests and resources are involved. If more requests are satisfied locally, system performance bottlenecks could be avoided at heads of the agent hierarchy and sub-hierarchies.

## Acknowledgements

The author would like to express his gratitude to members of the High Performance Systems Group at the University of Warwick, including Prof. Graham R. Nudd, Dr. Darren J. Kerbyson, Dr. Stephen A. Jarvis, Mr. Daniel P. Spooner and Mr. James D. Turner, for their previous contribution to the ARMS and ARMSim development.

## References

[1] V. S. Adve, R. Bagrodia, J. C. Browne, E. Deelman, et. al., POEMS: end-to-end performance design of large parallel adaptive computational systems, IEEE Transactions on Software Engineering 26(11) (2000) 1027-1048.

[2] K. Amold, B. O'Sullivan, R. Scheifer, J. Waldo, and A. Woolrath, *The Jini™ Specification* (Addison Wesley, 1999).

[3] F. Berman, A. J. G. Hey, and G. Fox, *Grid Computing: Making The Global Infrastructure a Reality* (John Wiley & Sons, 2003).

[4] R. Buyya, and M. Murshed, GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency and Computation: Practice and Experience, 14(13-15) (2002) 1175-1220.

[5] J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, Performance modelling of parallel and distributed computing using PACE, in: *Proceedings of 19th IEEE International Performance, Computing and Communication Conference (IPCCC '00)* (Phoenix, AZ, USA, 2000) pp. 485-492.

[6] J. Cao, D. J. Kerbyson, and G. R. Nudd, High performance service discovery in large-scale multi-agent and mobile-agent systems, International Journal of Software Engineering and Knowledge Engineering (Special Issue on Multi-Agent Systems and Mobile Agents) 11(5) (2001) 621-641.

[7] J. Cao, D. J. Kerbyson, and G. R. Nudd, Performance evaluation of an agent-based resource management infrastructure for grid computing, in: *Proceedings of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '01)* (Brisbane, Australia, 2001) pp. 311-318.

[8] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd, ARMS: an agent-based resource management system for grid computing, Scientific Programming (Special Issue on Grid Computing) 10(2) (2002) 135-148.

[9] J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini, and G. R. Nudd, Agent-based grid load balancing using performance-driven task scheduling, in: *Proceedings of 17th IEEE International Parallel and Distributed Processing Symposium (IPDPS '03)* (Nice, France, 2003) pp. 49-58.

[10] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, GridFlow: workflow management for grid computing, in: *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '03)* (Tokyo, Japan, 2003) pp. 198-205.

[11] H. Casanova, Simgrid: a toolkit for the simulation of application scheduling, in: *Proceedings of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '01)* (Brisbane, Australia, 2001) pp. 430-437.

[12] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, Web services description language (WSDL) 1.1 (W3C Note, 2001). http://www.w3c.org/TR/wsdl.

[13] I. Foster and C. Kesselman, Globus: a metacomputing infrastructure toolkit, International Journal of High Performance Computing Applications 2 (1997) 115-128.

[14] I. Foster and C. Kesselman, *The GRID: Blueprint for a New Computing Infrastructure* (Morgan-Kaufmann, 1998).

[15] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, PACE – a toolset for the performance prediction of parallel and distributed systems, International Journal of High Performance Computing Applications (Special Issue on Performance Modelling – Part I) 14(3) (2000) 228-251.

[16] D. Slama, J. Garbis, and P. Russell, *Enterprise Corba* (Prentice Hall, 1999).

[17] H. J. Song X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, The MicroGrid: a scientific tool for modeling computational grids, in: *Proceedings of Supercomputing 2000 (SC '00).*

[18] D. P. Spooner, S. A. Jarvis, J. Cao, S. Saini and G. R. Nudd, Local grid scheduling techniques using performance prediction, IEE Proceedings - Computers and Digital Techniques 150(2) (2003) 87-96.

[19] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima, Overview of a performance evaluation system for global computing scheduling algorithms, in: *Proceedings of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8)* (1999) pp. 97-104.