

TEMPLATES FOR DISTRIBUTED AGENT-BASED SIMULATIONS ON A QUASI-OPPORTUNISTIC GRID

Laszlo Gulyas^(a,b), Walter de Back^(b), Gabor Szemes^(b,a),
Krzysztof Kurowski^(c), Werner Dubitzky^(d), George Kampis^(b)

^(a)AITIA International Inc, Budapest, Hungary

^(b)Collegium Budapest, Hungary

^(c)University of Queensland, Brisbane, Australia

^(d)University of Ulster, Ireland

lgulyas@aitia.ai, wdeback@colbud.hu, gszemes@aitia.ai,
k.kurowski@imb.uq.edu.au, w.dubitzky@ulster.ac.uk, gkampus@colbud.hu

ABSTRACT

Complex systems are defined as systems with many interdependent parts which give rise to non-linear and emergent properties. Supercomputers constitute the *de facto* technology to deliver the required computational performance. However, supercomputers involve considerable costs, which many organizations cannot afford. The working assumption of this paper is that a *grid* could be enhanced by suitable middleware to provide features similar to those of supercomputers. However, simulation developers will face additional difficulties when adopting their applications to the grid. That is because the underlying topology of the computational infrastructure is dynamic. This paper reports on an ongoing effort to develop templates for distributed simulations on the grid with the integration of the Repast and ProActive packages.

Keywords: grid computing, agent-based simulation, opportunistic supercomputing, complex systems simulation templates

1. INTRODUCTION

Complex systems are defined as systems with many interdependent parts which give rise to non-linear and emergent properties determining the high-level functioning and behavior of such systems. Due to the interdependence of their constituent elements and other characteristics of complex systems, it is difficult to predict system behavior based on the ‘sum of their parts’ alone. Examples of complex systems include bee hives, bees themselves, human economies and societies, nervous systems, molecular interactions, cells and living things, ecosystems, as well as modern energy or telecommunication infrastructures. Arguably one of the most striking properties of complex systems is that conventional experimental and engineering approaches are inadequate to capture and predict the behavior of such systems.

To complement the conventional experimental and engineering approaches, computer-based simulations of complex natural phenomena and complex man-made artifacts are increasingly employed across a wide range of sectors. Agent-based simulation is a suitable and useful modeling paradigm for the decomposition and study of complex systems. (North and Macal, 2007)

Typically, such simulations require computing environments which meet very high specifications in terms of processing units, primary and secondary storage, and communication. Supercomputers constitute the *de facto* technology to deliver the required specifications. Acquiring, operating and maintaining supercomputers involve considerable costs, which many organizations cannot afford. The working assumption of this paper (following that of the QosCosGrid project (Coti et al, 2008), <http://www.qoscogrid.eu/>) is that a *grid* could be enhanced by suitable middleware to provide features and performance characteristics that resemble those of a supercomputer. We refer to such a grid as *quasi-opportunistic supercomputer*. The QosCosGrid project aims at developing such a system.

Computational simulations on supercomputers or on grid systems naturally require a *distributed* implementation. However, this complicates the model development significantly, especially, in cases of experimental, incremental model development, where the model structure may change dramatically during development. Moreover, the implementation complexity may go beyond the capabilities or interests of researchers in complex systems.

To face these issues, we are developing integration between the agent simulation toolkit Repast (North et al, 2006) and ProActive (Baude et al, 2006), a middleware for multi-threading, parallel and distributed computing, currently being interfaced with the QosCosGrid. This combination facilitates the design and deployment of distributed agent-based simulations over multiple computational nodes on multi-core machines, local clusters, and grid environments.

Naturally, the distribution and parallelization of computer programs, and thus simulations cannot be fully automatized. Parallelization is a multi-dimensional optimization problem. Two of the major dimensions are the minimization of communication between nodes, and the balancing of the processing load among the nodes. Except for the simplest programs, these are non-trivial issues that require in-depth knowledge about the workings of the particular program (simulation). However, general schemas for solutions exist, especially if the priority among the two major dimensions above is defined.

Our approach is to focus on the *minimization of communication*. We have identified classes of simulations that share common (agent-to-agent) communication templates. We are currently in the process of developing distributed and parallel Repast/ProActive implementations for these templates that can be subclassed and customized for particular user simulations. (Load balancing and other issues are handled, optionally, within the framework of the selected template.) The supported templates range from embarrassingly parallel applications such as parameter sweeps, to cellular automata, to static and dynamic (communication) networks, and to agents moving in abstract spaces.

The paper overviews the project's main directions and presents the current status, including working examples and prototypes.

2. THE QOSCOSGRID

The term *supercomputer* typically refers to a dedicated special-purpose multiprocessor computing system that provides close to best achievable performance for demanding parallel workloads. Supercomputers have several characteristics that enable them to efficiently execute considerable computational loads.

1. High-end and highly reliably hardware components, such as processing units, primary and secondary memory, and interconnects
2. Supercomputer middleware provides a straightforward abstraction of a homogeneous computational and networking environment, automatically allocating resources according to the underlying networking topology
3. The resources of a supercomputer are managed exclusively by a single centralized system, which enforces global resource utilization policies, thus maximizing hardware utilization while minimizing the turnaround time of individual applications.

These characteristics endow supercomputers with unprecedented performance, stability, and dependability properties.

Grid computing systems could be viewed as large-scale computing systems with considerable levels of hardware resources but with a lack of the features that make supercomputers so powerful. In particular, grids usually lack sophisticated support for highly parallel applications with significant inter-process communication requirements. Grid computing environments are based on heterogeneous, widely dispersed and time-variant resources which typically lack central control. Connected via local and wide area networks, grids typically rely on an opportunistic

marshaling of resources into coordinated action to meet the needs of large-scale computing applications. Grids are often offered as panacea for all kinds of computing applications, including those that require supercomputing-like computing environments. However, this vision of grids as virtual supercomputers is unattainable without overcoming the performance and reliability issues plaguing current grids.

The main challenge of the QosCosGrid project is to overcome the current limitations of grids and implement a virtual computer which could be considered a viable approximation of a real supercomputer. The details of the proposed technical solutions are reported elsewhere. (Coti et al, 2008)

2.1. Requirements of Complex Systems Simulations

Despite the variety of scientific, engineering and other areas in which complex systems need to be studied and modeled, the key information technology requirements for computational modeling and simulation of complex systems are essentially identical across many domains and applications. These requirements include:

- Integration of large heterogeneous volumes of data and information that may arise from simulation or other information systems, which may be geographically widely dispersed.
- Design and execution of compute- and memory-intensive simulation programs may require resources that are not available locally.
- Handling the considerable volumes of data generated as output from the underlying simulations. These data need to be managed, analyzed and then shared using varied computational methodologies.

To address these requirements and achieve the main aims of the QosCosGrid project, the project is structured into the following key objectives:

- First, to provide a quasi-opportunistic supercomputing grid architecture and infrastructure which includes (i) Necessary grid middleware services including monitoring and measurement capabilities, (ii) User interfaces that enable easy access and use of resources by hiding the underlying complexity of the system, (iii) Flexible fault-tolerant message passing libraries, (iv) Data distribution enabling technology, and (v) Remote steering capabilities;
- Second, to develop services that provide (i) Dynamic resource brokering giving the best quality-of-service to any given complex system simulation, (ii) Reservation and orchestration of resources, communication, synchronization and routing as known from massively parallel processors computers.
- Third, to validate the quasi-opportunistic supercomputing concept with various types of complex systems simulation applications including (i) Research into the non-trivial parallelization of the simulation- and data-

processing applications typically encountered in the CS research, and (ii) to adapt the underlying algorithms to the quasi-opportunistic supercomputing environment.

This paper is focused on the last two items of the above ambitious set of goals.

3. COMPLEX SYSTEMS SIMULATIONS ON THE GRID

In this section, we provide a general abstract description of complex systems simulations that captures essential properties influencing the distributed implementation of such systems. As with every abstraction, certain details of individual cases are omitted, but to our belief, without the loss of generality.

(1) Interaction topology. A complex system consists of a finite set of interacting components. In our abstract treatment, each component will have a single state variable and an update function. (Notice that the single state variable can, in practice, be a combination of any finite number of variables.) The update function depends on the state variable itself and on the states of a subset of the other components. (The update function can be deterministic or probabilistic, in which latter case it yields a probability distribution of the next state.) The update functions' dependence on other components defines the *communication pattern* or *interaction topology* of the complex system.

(2) Parameter space. Simulations deal with the calculation of the time-dynamics of the given complex system for a specific combination of initial parameters. In some cases, the execution of the simulation for a few initial parameter combinations suffices, but typically, a *parameter space search* is necessary to assess the system's behaviour over a range of parameter combinations.

(3) Homogeneity vs heterogeneity. Complex systems can be homogenous or heterogeneous, in that the components may or may not share the same type of state variables. Similarly, the communication pattern can also be non-uniform, when the dependencies of the update functions are heterogeneous. Additional complexity may result from the number of components changing in time, albeit in a theoretical discussion this can always be circumvented by giving an estimate of the maximum number of components. In some cases, the communication pattern is dynamic, even among a static set of components. Formally, this issue can also be avoided by assuming dependence on all other components, some of which dependences may be rendered temporarily inactive. However, our goal is to exploit as much of the available dependency information as possible, therefore, we will explicitly deal with the temporal dynamics of the interaction topology when necessary.

(4) In-run and inter-run parallelization. For exploiting the benefits of a distributed, parallel implementation two major strategies exist. One distributes entire simulation runs across a pool of

computers (termed in various ways: *parameter space search*, *parameter sweeping*, *inter-run parallelization*, etc.), while the other attempts to distribute individual runs across computers (sometimes termed *in-run*, or *intra-run parallelization*). It is worth noting that in many cases complex systems simulations are communication heavy, implying that their distribution incurs a strong communication penalty. Yet, in some cases, it is still worthwhile to distribute them for memory gain: a distributed implementation allows for experimenting with large systems (in the number of components) that would otherwise be impossible or very costly.

In the following discussion, we will treat in-run and inter-run parallelization in a unified discussion. Our goal in doing so is to provide a short check-list and a characterization that helps in identifying the potential benefits of a distributed implementation for any given complex systems simulation.

3.1. Partitioning Complex Systems Simulations

The main challenge addressed here is the allocation of complex systems components to computational nodes, i.e., the partitioning of the complex systems simulation, subject to minimizing the execution time of the distributed implementation. (Note, that a version of the minimizing requirement is present also in the case when the goal of distribution is the memory gain.) To cut down on execution time in a distributed environment, one must

- *Minimize communication* among components in different partitions, and
- *Balance the computational load* at each partition (i.e., execution time in between information exchange among the partitions).

In order to achieve the latter, the computational activities at each computational node should be in proportion with the performance of that node. Grid systems are inherently heterogeneous and dynamic and, as discussed earlier, complex systems components can also be heterogeneous and dynamic also. These features taken together would make any abstract analysis very hard. Therefore, we decided to focus on the communication aspect of distribution, trying to classify complex systems simulations based on their communication patterns.

A simple way of putting this is formulating the *sufficient* assumption, that all computational nodes are uniform and homogenous. However, this is not a *necessary* requirement, our classification of communication patterns below can be also used to develop distributed simulations for non-uniform computational systems as well. In fact, more sophisticated implementations based on these classifications are expected to take load-balancing aspects into account as well. On the other hand, in grid systems, communication costs are often prohibiting across computational clusters, while they are more relaxed among computers belonging to the same cluster. Therefore, our focus on minimizing communication is

also useful to allocate complex systems components to computational clusters, assuming that load balancing will be handled locally, at the local clusters.

3.2. Communication Templates for Complex Systems Simulations

Our approach here is to study the interaction topology of complex systems simulations in isolation, in an attempt to classify frequently occurring cases into communication *templates*. Each identified template can then be accompanied by template implementations and usage advices, based on the classic parallel and distributed computing literature. The general idea here is that complex system modelers can (i) first identify the communication class their models belong to, and then (ii) “fill in” one of the implemented simulation templates provided with their model-specific details. This way, they might not achieve the most efficient distributed implementation of their models, but their implementation efforts will be significantly lowered.

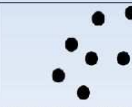





We first point out two extreme cases. In the first scenario, dependencies among components are completely random (e.g., from a uniform random distribution) and change regularly in time (e.g., dependencies are re-sampled prior to each update). The second special case is when no communication occurs among complex systems components. One might argue that these are “useless” or “hopeless” examples, and that a collection of components like in our second example is hardly a *system* at all. However, we believe this is but a question of level of abstraction. In the first scenario, which is not as uncommon as it may seem, there is hardly a better advice regarding a distributed implementation than relying on a distributed parameter space search. Which is just an example of our second scenario: the individual simulation runs can be viewed as non-interacting components. On the other hand, more sophisticated parameter space search methods introduce dependencies among individual runs, by determining the next parameter combinations to explore based on results collected earlier (i.e., sampling in the more turbulent parameter regions, etc.). In this case, the parameter space search becomes a non-trivial complex system again, worthy of dependency analysis on its own.

Dealing with more complex cases, our first observation is that *static* communication patterns allow for the direct application of distribution algorithms. Therefore we will handle these cases separately from the *dynamic topologies*.

Next, we point out that the dependencies of the update functions may be dependent on the components’ states. In many cases, the components’ state information can be projected to a metric space and update dependencies correlate with distance in this space. For example, if components are agents moving in a space (in computational models often on a two-dimensional lattice) then, each agent’s state will (among possibly other things) include the coordinates of the agent. If in the model the agents interact only with the agents in

their vicinity, then the update dependencies of the components will be thus distance-dependent. This *spatial property* of a complex systems simulation, if present, may be successfully exploited in determining the partitions of a distributed implementation. It is worth noting, however, that complex systems may have such a *spatial property* implicitly. For example, a social system where people are likely to interact with like-minded partners may have this property where the natural metric space is an abstract similarity space.

Table 1. Categorization of the Various Communication Templates

Categorization		
Categorization in various templates (T) based on CSS structure:		
Special Cases	 Independent Components T0	 Unknown Topology T5
	Static	Dynamic
Network	 Fixed graph T1	 Dynamic graph T2
Spatial	 Cellular Automata T3	 Mobile Agents T4

Based on the observations above, we propose 5+1 *communication templates* that, as we believe, are the commonly occurring classes of complex systems simulations.

Certainly, it would be possible to identify many more, or to refine this classification by dividing some of the templates proposed here. However, we believe these 5+1 patterns¹ are applicable (see Table 1), to a varying extent, a large majority of complex systems simulations, and that useful advices can be formulated for each of them.

Template0 (T0) of our classification is the case where no interaction occurs among components. Here the components’ partitioning is only constrained by load balancing considerations.

Template5 (T5), the other extreme, is the case with random or unpredictable interaction among components, for which we suggest ‘to step one level up’ and the implementation of a distributed parameter search as in Template0. The four remaining cases are created at the intersection of the spatial/non-spatial and static/dynamic properties.

¹ We identify 6 communication topologies (or templates), but since for the two extreme cases (T0 and T5) our proposed technical solution is very similar, albeit for different reasons, we prefer to talk about 5+1 templates.

Template1 (T1) (*Static Networks*) describes a non-spatial system with a static communication pattern. It is assumed that the exact communication pattern can be extracted from the system, or that it is defined explicitly. To the thus defined *communication graph* a variety of *graph partitioning algorithms* can be applied. (Fjällström, 1998)

Template2 (T2) (*Dynamic Networks*) introduces dynamism in Template1. The assumption about the existence of a communication *graph* is maintained, but, in contrast to Template5, it is assumed that the changes and their frequency are defined by a graph transition function that provides enough information on the system's efficient distribution. (Sometimes, it may be sufficient to know that the level of change in the communication graph is low, such that it is sufficient to re-partition nodes at every 10000 time steps, or so.) The implementation approach we propose for such systems is the regular application of classic *graph repartitioning algorithms* (i.e., graph partitioning algorithms that attempt to improve on an existing partition). (Barnard and Simon, 1993) Please also note, however, that in the limiting case, Template2 leads to Template5.

Template3 (T3) (*Static Spatial Systems*) moves away from Template1 along the other axis. It maintains the assumption about a static communication pattern, but requires the *spatial property*. Prime examples of such systems are *cellular automata*. Template3 is the pattern most prone to a distributed implementation, most of which can be derived from methods and algorithms developed for distributed cellular automata implementations. (Mazzariol et al, 2000)(Maniatty et al, 1998)

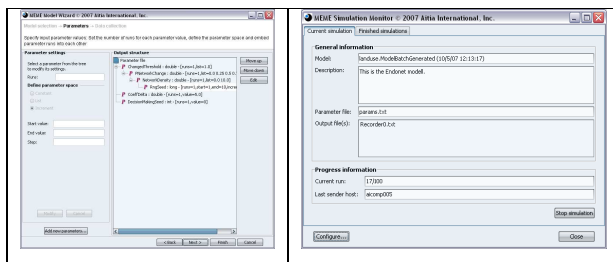


Fig. 1. The Pages from the Distributed Parameter Sweep Wizard for Repast

Finally, **Template4 (T4)** (*Dynamic Spatial Systems*) assumes a spatial system, in which the communication pattern evolves over time. One example of such system is the above discussed case when agents move in space and communicate with those in their vicinity. For the distribution of complex systems belonging to this template, we point to algorithms specially developed for such systems, using buffering and messaging solutions and ways of predicting the speed of spatial movement. (Scheutz and Schermerhorn, 2005)(Gilbert et al, 2006) (The latter may be used to predict the next time communication among partitions or repartitioning becomes necessary.) It is worth pointing out that the fundamental assumption of this

template and thus a key to the successful implementation of these solutions is that changes in spatial positions are slow relative to the frequency of state updates.

3.3. Implementation Status

Above we have categorized complex systems simulations into 5+1 classes, based on their internal communication structure. These *communication templates* form the base of the *simulation templates* written in Repast that QosCosGrid will provide for complex systems modelers.

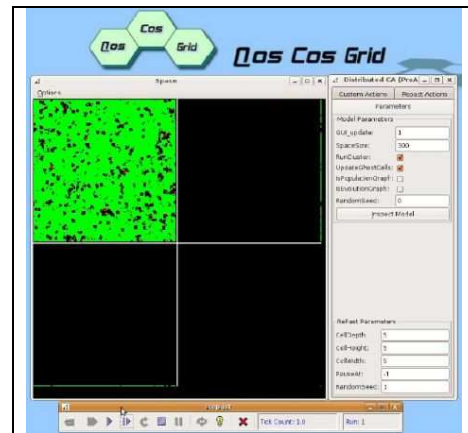


Fig. 2. Screenshot from a Distributed Cellular Automata Simulation Implemented with the T3 Simulation Template.

The project reported in this paper is work in progress. A number of working results are already available. An advanced tool for user friendly and extensible parameter sweeping (a solution offered for T0 and T5 above) of Repast simulations with ProActive was reported in (Iványi et al, 2007) and is available from <http://meme.aitia.ai/>. (See Fig. 1.) Furthermore, simulation templates for models with complying with T1 or T3 were reported in (de Back et al, 2008). (See Fig. 2.) The development of simulation templates for T2 and T4 is an ongoing effort.

4. DISCUSSION AND RELATED WORK

Distributed programming is almost as old as computer science itself. Therefore the problem is *graph partitioning* (which is the theoretical abstraction of communication minimization and load balancing) is an aged and well-studied problem. (Fjällström , 1998) In fact, the simulation templates proposed above for T1-T4 are all based on decades old algorithms, except for one, T4. (Barnard and Simon, 1993) (Hendrickson and Leland, 1995)(Karypis and Kumar, 1997)(Ou and Ranka, 1997)

On the other hand, complex systems simulations have been implemented for supercomputers for several decades now (depending on the exact definition of the field). However, these implementations were mostly done by professional computer scientists and were

tailored to the particular problem and infrastructure at hand. (That is, naturally, the approach to take for the most efficient implementation.)

Over the last decade, the increasing capacity of computers and the advent of the agent-based modeling approach made possible to describe models in a way close to the modeled system. This development opened the way for a whole new class of scientists to create and study their models without relying on professional programmers. However, a distributed implementation was so far beyond the reach of these modelers. Work only recently started to address this problem. (Chen et al, 2008) The simulation templates proposed in this paper belong to these novel efforts. They are intended to be used as a blueprint by complex systems modelers.

5. CONCLUSIONS

This paper reported on an ongoing effort to develop templates for distributed simulations on the grid with the integration of the Repast and ProActive packages. The approach to manage the multi-dimensional optimization problem of program partitioning was discussed, together with a classification of complex systems simulations into 5+1 categories, based on their communication structure. These communication templates are then used to develop implementation schemas for complex systems simulations.

The general context of is a *quasi-opportunistic* approach (i) to develop a special purpose middleware that augments grid systems with supercomputer-like properties (number of processing units, temporarily static communication channels, etc.), and (ii) to provide complex systems modelers with a flexible and easy-to-approach platform to develop distributed agent-based simulations.

ACKNOWLEDGMENTS

The present work is being carried out within the frame of the QosCosGrid (Quasi-Opportunistic Supercomputing for Complex Systems Simulations on the Grid) project funded by the European Commission's 6th Framework Programme. The partial support of the EC under grant *QosCosGrid* IST FP6 #033883 is gratefully acknowledged.

REFERENCES

de Back, W., Szemes, G., Kampis, G., Gulyás, L., „Distributed simulation templates for repast”, In *Conference of the Swarm Development Group*, Chicago (SwarmFest 2008), 2008.

Barnard, S. T., & Simon, H. D., „A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems”, In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing* (pp. 711-718), 1993.

Baude F., Baduel L., Caromel D., Contes A., Huet F., Morel M. and Quilici R., [‘Programming, Composing, Deploying for the Grid’](#), in *GRID COMPUTING: Software Environments*

and Tools, Jose C. Cunha and Omer F. Rana (Eds), Springer Verlag, January 2006.

Chen, D., Theodoropoulos, G. K., Turner, S. J., Cai, W., Minson, R., and Zhang, Y., „Large scale agent-based simulation on the grid”, *Future Generation. Computer Systems*, 24, 7 (Jul. 2008), 658-671.

Coti, C., Herault, T., Peyronnet, S., Rezmerita, A., Cappello, F., „Grid Services For MPI”, *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08)*, to appear, Lyon, France, 2008. 05. 14.

Fjällström, P.-O., „Algorithms for Graph Partitioning: A Survey”, *Linköping Electronic Articles in Computer and Information Science*, 3, <http://www.ep.liu.se/ea/cis/1998/1010/>, 1998.

Gilbert, N., den Besten, M., Bontovics, A., Craenen, B. G. W., Divina, F., Eiben, A. E., et al., „Emerging Artificial Societies Through Learning”, *Journal of Artificial Societies and Social Simulation* 9 <http://jasss.soc.surrey.ac.uk/9/2/> 9.html, 2006.

Hendrickson, B., & Leland, R., „An improved spectral graph partitioning algorithm for mapping parallel computations”, *SIAM J. Sci. Comput.*, 16, 452-469., 1995.

Iványi, M., Gulyás, L., Bocsi, R., Szemes, G., Mészáros, R., „Model Exporation Module”, *Agent 2007: Complex Interaction and Social Emergence Conference*, Evanston, IL, November 15-18, 2007

Karypis, G., & Kumar, V., „A coarse grain parallel formulation of multilevel k-way graph partitioning algorithm”, In *Proc. Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

Maniatty, W. A., Szymanski, B. K., & Caraco, T., „Parallel computing with generalized cellular automata”, *Parallel and Distributed Computing Practices*, 1, 31-50., 1998.

Mazzariol, M., Gennart, B. A., & Hersch, R. D., „Dynamic load balancing of parallel cellular automata”. Paper presented at the *SPIE Conference: Parallel and Distributed Methods for Image Processing IV*, San Diego, USA, 2000.

North, M.J., Macal, C.M., *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation* (Oxford 2007)

North, M.J., N.T. Collier, and J.R. Vos, “Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit”, *ACM Transactions on Modeling and Computer Simulation*, Vol. 16, Issue 1, pp. 1-25, ACM, New York, New York, USA (January 2006).

Ou, C.-W., & Ranka, S., „Parallel incremental graph partitioning”, *IEEE Transactions on Parallel and Distributed Systems*, 8, 884-896, 1997.

Scheutz, M., & Schermerhorn, P., “Adaptive Algorithms for the Dynamic Distribution and Parallel Execution of Agent-Based Models”. *Journal of Parallel and Distributed Computing*, 66, 1037-1051., 2005.